Computing McGraw-Hill

DISK Included

GAME GRAPHICS IN C++

LEN DORFMAN

Game Graphics in C++

Other books by Len Dorfman

C++ by Example: Object-Oriented Analysis, Design, & Programming *by Len Dorfman*

Lightning-Fast Animation Graphics *by Len Dorfman*

Other McGraw-Hill titles of interest

Advanced Multimedia Programming *by Steve Rimmer*

Bitmapped Graphics *by Steve Rimmer*

C for Windows Animation Programming *by Lee C. Adams*

C Mathematical Function Handbook *by Louis Baker*

C Primer (The) *by L. Hancock*

C Tools for Scientists and Engineers *by Louis Baker*

C + C++: Programming with Objects in C and C++ *by A. I. Holub*

C++ and the OOP Paradigm *by B. R. Rao*

C++ Applications Guide *by D. N. Smith*

C++ Memory Management *by Len Dorfman, Marc J. Neuberger*

C++ Power Paradigms *by Mark Watson*

C++ Primer for C Programmers *by Jay Ranade, Saba Zamir*

Canned Code for DOS and Windows *by Steve Rimmer*

Concepts of Object-Oriented Programming *by D. N. Smith*

Constructing Windows Dialogs *by Steve Rimmer*

Creating Artificial Life: Self-Organization *by Ed Rietman*

Elements of C Programming Style (The) *by Jay Ranade*

Garbage In, Gorgeous Out: Text and Graphics Translation *by Walt Garnett*

Genesis Redux: Experiments Creating Artificial Life *by Ed Rietman*

Graphics File Formats *by David C. Kay, John R. Levine*

Graphics Programming Solutions *by Julio Sanchez, Maria Canton*

High-Resolution Video Graphics *by Julio Sanchez, Maria Canton*

McGraw-Hill Multimedia Handbook (The) *by Jessica Keyes*

More C Tools for Scientists and Engineers *by Louis Baker*

Multimedia Applications Development *by Mark J. Bunzel, Sandra K. Morris*

Multimedia Programming for Windows *by Steve Rimmer*

Portable User Interface Programming in C++ *by Mark Watson*

Practical C++ by *M. A. Terribile*

Super VGA Graphics Programming Secrets *by Steve Rimmer*

Visual C++ Generic Programming *by Namir C. Shammas*

Windows Visualization Programming with C/C++ *by Lee C. Adams*

Game Graphics in C++

Len Dorfman

McGraw-Hill, Inc.

New York San Francisco Washington, D.C. Auckland Bogotá
Caracas Lisbon London Madrid Mexico City Milan
Montreal New Delhi San Juan Singapore
Sydney Tokyo Toronto

©1995 by **McGraw-Hill, Inc.**

Printed in the United States of America. All rights reserved. The publisher takes no responsibility for the use of any materials or methods described in this book, nor for the products thereof.

pbk 1 2 3 4 5 6 7 8 9 0 DOC/DOC 9 9 8 7 6 5

Product or brand names used in this book may be trade names or trademarks. Where we believe that there may be proprietary claims to such trade names or trademarks, the name has been used with an initial capital or it has been capitalized in the style used by the name claimant. Regardless of the capitalization used, all such names have been used in an editorial manner without any intent to convey endorsement of or other affiliation with the name claimant. Neither the author nor the publisher intends to express any judgment as to the validity or legal status of any such proprietary claims.

Library of Congress Cataloging-in-Publication Data
Dorfman, Len.
Game graphics in C++ / by Len Dorfman.
p. cm.
Includes index.
ISBN 0-07-911951-4 (p)
1. Computer graphics. 2. Computer animation. 3. C++ (Computer program language) 4. Computer games. I. Title.
T385.D677 1995
794.8'15133—dc20 95-9681
CIP

Acquisitions editor: Jennifer Holt DiGiovanna
Editorial team: Kellie Hagan, Book Editor
Robert E. Ostrander, Executive Editor
Production team: Katherine G. Brown, Director
Rose McFarland, Desktop Operator
Lorie L. White, Proofreading
Jodi L. Tyler, Indexer
Design team: Jaclyn J. Boone, Designer
Katherine Stefanski, Associate Designer

WK2
9119514

Contents

Acknowledgments *ix*

Introduction *xi*

Part 1
The demonstration programs

1 Sprite, missile, and bullet objects *3*

What are bit planes? *4*
What is bit-plane animation? *6*
How is bit-plane animation accomplished? *6*
Designing sprites *8*
Summary *8*

2 Moving a sprite object in a playfield space *11*

Demonstration of a seamless interface to Borland's GI *12*
Including the sprite object file *13*
Creating an instance of the sprite class *13*
Creating an instance of the playfield class *14*
Moving the sprite instance on the playfield *14*
Destroying the playfield *14*
Summary *15*

3 Moving a three-colored sprite *47*
Moving a multicolored sprite on the graphics playfield *49*
Summary *50*

4 A within-sprite animation technique *79*
Changing a multicolored sprite shape as direction changes *80*
Summary *81*

5 Simultaneous sprite, missile, and bullet movement *131*
Polling *132*
Sprite, missile, and bullet movement *133*
Summary *134*

6 Controlling a sprite within a track *147*
Keeping the sprite in line *148*
Summary *149*

7 Launching a missile from a moving sprite *215*
Launching a missile *216*
Summary *216*

8 A simple moving-tank-shoots-chasing-ghosts game *227*
The tank game framework *228*
Detecting ghost (ghost collisions) *229*
Detecting a ghost (missile collisions) *229*
Summary *229*

Part 2
Source code for the bit-plane animation class library

9 Assembly source code for the sprite library *269*
Assembly sprite management functions *270*
Summary *270*

10 C++ source code for the bit-plane animation class library *275*
Source code for the sprite class *276*
Source code for the missile class *276*
Source code for the bullet class *276*

Source code for the video class *276*
Source code for the playfield class *277*
Source code for the key class *277*
Source code for the sound class *277*
Source code for GO.CPP *277*
Summary *277*

Part 3
The sprite editor

11 The sprite editor *305*
How to use the sprite editor *306*
Sprite editor C source code *309*
Summary *309*

12 C language sprite editor support functions *337*
Support function C source code *338*
Summary *339*

13 Assembly language sprite editor support functions *357*
Support function assembly language support code *358*
Summary *359*

Epilogue *377*

Index *379*

About the author *383*

About the disk *386*

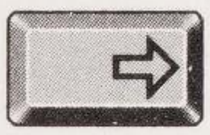
Acknowledgments

I'd like to thank my acquisitions editor, Jennifer Holt DiGiovanna, for her help in drawing from my stream of off-beat ideas and turning them into book proposals.

Also, heart-centered thanks to Barbara, my wife of a quarter century, and Rachel, our daughter, for their continuous expressions of loving kindness and unwavering support of playtime.

Introduction

I wrote this book and its C companion, *Lightning Fast Animation Graphics*, for fun. Since my old Atari 800 assembly programming days, I've enjoyed both playing and writing "classic" video games in the Pac-Man and Lode Runner genre—not for profit, but for fun. And it's in that spirit that this book was born.



The purpose of this book

The purpose of this book is to provide both C programmers who want to wade in C++ waters and C++ programmers a C++ class library that greatly simplifies the task of creating bit-plane animation effects in DOS-based programs. This book supports the development of three types of graphic objects: sprites (in this book, a 47 × 46, multicolored, programmer-defined graphic object), missiles, and bullets.

The C++ class libraries contain class methods that instantiate graphic objects, smoothly move them within a bit plane, and detect collisions between sprite, missile, and bullet objects. These classes provide you with reusable tools that will prove effective in creating special graphics effects and shoot-'em-up video games.

The creation of sprite object definition data is greatly simplified by the sprite editor program included in this book and its companion disk. The editor lets you design the multicolored sprite shape using a mouse, and then permits you to save the sprite's shape back to disk. Although the sprite editor is written in hybrid C and assembly, the sprite definition file that's written to disk by the sprite editor is in standard C format.

How this book is organized

Part 1 is designed for the applications programmer. It begins by introducing the concept of sprites, missiles, and bullets in the context of 640 × 480, 16-color, VGA graphics. Heavily documented demonstration programs follow, which explain computer-controlled sprite, missile, and bullet motion in the context of polling techniques. Then you'll see a method of controlling sprite motion from the keyboard, followed by sprite, missile, and bullet collision detection. Part 1 ends by presenting the source for an in-progress chase-in-maze game.

Part 2 contains the C++ graphic object source code for the sprite, bullet, and missile classes. I've been careful to craft these classes in C++, so if you aren't an assembly wizard you'll still be able to see how the code works and easily modify it. I chose not to present the assembly versions of these routines because, in testing, I couldn't find any real-world difference in animation performance between the assembly and C++ class library modules.

Part 3 contains the C and assembly source code for the sprite object editor. This editor greatly simplifies the design of sprites. To design a multicolored sprite, simply load the program, use the cursor to define the object of the sprite, and then save the sprite object back to disk. You can include the sprite object C source file in your program, or you can compile or add it to a sprite object library.

Note that bit-plane animation is only one of many different ways of producing animation on a PC. This book isn't an animation reference manual; rather, it's specifically designed to facilitate the inclusion of

high-speed sprite, missile, and bullet movements into your reusable C++ class library programming arsenal. Also note that the listings are located at the end of each chapter.

What you need to use this book

You'll need a computer that supports the VGA, 640 × 480, 16-color video mode. For this book I've used Borland's C/C++ versions 3.1 and 4.0 compilers and TASM assembler. I've designed the C++ class source to be well behaved, and it should easily compile with other C++ compilers.

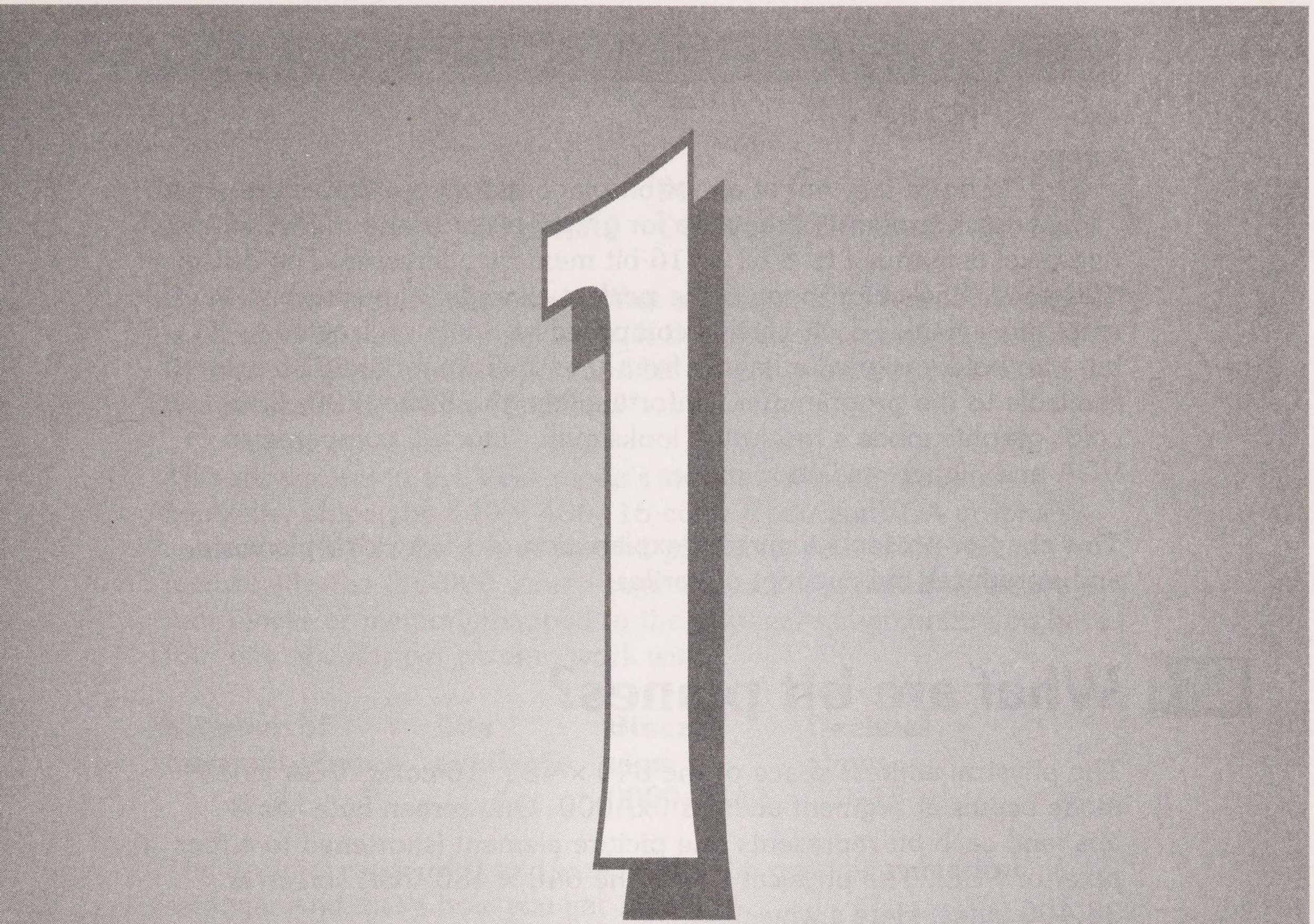
Note, however, that I've integrated the bit-plane animation programs with Borland's graphic interface (BGI), so if you're using another compiler you'll need to use that compiler's graphic interface. The sprite code, however, should run as is. Once the graphic mode is set to VGA, 640 × 480, 16-color mode, the sprite code will fly.

Part 1

The demonstration programs

Part one contains demonstration programs that progressively demonstrate the use of sprite, missile, and bullet graphic objects in practical programs. As you work your way through this part of the book, you'll learn how to integrate the powerful graphics methods presented in Part 2 to create animation displays in your VGA-based programs. Feel free to use portions of these programs in any commercial or private programs you write.

2



11

Sprite, missile, and bullet objects

CHAPTER 1

I'M not a fan, not at all, of bit-plane graphics architecture. I much prefer writing code for graphics cards and modes where one pixel is mapped to 8-bit or 16-bit memory addresses. The 320 × 200-pixel, 256-color mode is the perfect example of a sensible mapping scheme. Each pixel is composed of 8 bits or 1 byte. As 8 bits can hold unsigned numbers from 0 to 255, there are 256 colors available to the programmer. Unfortunately, the 320 × 200, 256-color graphic mode's resolution looks quite "blocky" compared to VGA and higher-resolution modes.

This chapter presents a cursory explanation of VGA video planes, and introduces the concept of sprites.

What are bit planes?

The physical address space of the 640 × 480, 16-color, VGA video mode begins at segment address 0xA000. One screen byte has 8 bits, and each bit represents one picture element (shortened to either *pixel* or *PEL*). The physical size of the 640 × 480 VGA screen is 38,400 bytes. Here's why:

Let's figure out how many bytes per screen row. If there are 640 pixels per row in the 640 × 480 mode, then you calculate the number of bytes by dividing 640 by 8. This equals 80, so there are 80 bytes per row in the 640 × 480 screen. There are 640 columns and 480 rows in the 640 × 480 VGA screen. You can calculate the size of the VGA physical space by simply multiplying the number of bytes per row (80) by the number of rows (480). The product of that multiplication is 38,400.

In a true sense, the physical memory starting at segment 0xA000 and continuing for 38,400 bytes is a binary representation of the screen. If address A000:0000 holds an 0xFF (all bits on), then the first eight pixels of the upper left-hand corner of the screen will be on. If address A000:0000 holds an 0xAA (every other bit on), then the first eight pixels of the upper left-hand corner of the screen will alternate, as follows:

Address	Hex	Bit	Pixel
A000:0000	0xAA	10101010	On-Off-On-Off-On-Off-On-Off

By now, I hope you're saying, "Wait a minute. If a physical screen bit is equal to 1, then the pixel is on. If the physical screen bit is equal to 0, then the pixel is off. Bit on and bit off are two bit states; how do two bit states turn into 16 colors?"

The answer lies in the VGA mode's architecture and graphics controller chip. The 640 × 480, 16-color mode the VGA graphics controller chip maps one of four bit planes (or four blocks of memory) to the 0xA000 screen segment address area. If there are four blocks of memory mapped to the physical screen area, you have four bits of color per pixel to work with:

Number of mapped planes	Bits available	Binary view	Decimal view
4	4	0000–1111	0–15

The decimal range of 0 to 15 allows for 16 possible numerical settings. And that's how you get 16 colors.

How many planes would you need to get, say, 256 colors? If four planes (four bits) translates to 16 colors, then eight planes (eight bits) would translate to 256 colors. How many planes would you need to display 1,024 colors? If you see why the answer is 10 planes, then you get it.

So a bit plane is a block of memory that's mapped to the screen by the VGA graphics controller chip, starting at memory segment 0xA000. The greater the number of bit planes you have available to be mapped to your physical screen memory, the more colors your VGA video adapter will support. That's why VGA video cards come with differing amounts of memory. The more memory on your VGA card, the greater the number of bit planes you'll have available to be mapped to memory and the more colors you'll have. Simple as that.

Now, is the VGA video bit-planing scheme the best way to map a graphics display to memory? Heck no! That, however, is what we're dealing with.

What is bit-plane animation?

Now that you understand what bit planes are, let's look at bit-plane animation. Animation, for the purposes of this book, means moving graphic objects on the screen. The bit-plane animation techniques used in this book permit you to move three types of fixed-size graphic objects very smoothly, at low and high speeds. These graphic object types are:

Sprite A 47 × 46 graphic object defined by the sprite editor.

Missile A small round ball object (fixed object and size).

Bullet A very small round ball object (fixed object and size).

So, for the purposes of this book, bit-plane animation means moving sprites, missiles, and bullets smoothly about the screen at varying speeds.

How is bit-plane animation accomplished?

The following is a simplified overview of how bit-plane animation is accomplished:

1. Select (or map) a VGA bit plane using the VGA graphics controller chip.
2. Write the graphic object (sprite, missile, bullet) directly to the appropriate address of physical screen memory.

The actual bit-plane animation process used in this book, however, is far more complicated. Although I'll be presenting details about bit-plane animation techniques, if you use the collection of sprite animation video functions in the text you don't need to be aware of the intricacies of bit-plane animation.

For the purposes of this book, the screen background over which the graphic objects are moved is called the *playfield*, and the method of writing the graphic object to the screen can be described as "writing the graphic object over the playfield." So moving a graphic object consists simply of writing the graphic object over the playfield.

The problem is that writing the graphic object over the playfield actually makes repeated images of the graphic visible, which isn't the desired effect. The technique used in this book to eliminate the problem is as follows:

❶ Write the graphic object to the playfield bit plane.

❷ Let the object display for a specified time (long enough for your eye to register it).

❸ Remove the graphic object from the screen.

❹ Rewrite the graphic object at a contiguous location in the desired direction.

OK. You now might ask, "How do you erase the graphic object without blotching the playfield?" Good question. Here's the answer:

There are a few ways to handle this graphic object erasure problem. I've chosen a simple and elegant method, but there's a trade-off. A bit-plane animation initialization routine that I wrote methodically saves all the screen memory of the four bit planes that hold the information for the playfield. When you erase a graphic object, you're really copying the original playfield data to the appropriate bit plane. Simple, huh?

Here's the trade-off. What happens if two graphic objects moving on the same plane overlap? There will be some object flutter. The faster the objects are moving, the less the eye will notice it. I could have eliminated this special condition by saving the playfield screen data to memory, drawing the sprite, then erasing the sprite by restoring the previously saved playfield object.

I didn't take that course. Here's why. When I was writing the demo programs, I wanted to have half a dozen or more objects moving smoothly about the screen at high speed. Repeatedly saving the

graphic object slowed things down. So I chose speed of graphic object motion over the potential object flutter under very specific and predictable conditions.

Designing sprites

You can design sprite objects in two ways. You can use the sprite editor provided on the companion disk, whose source code is presented in Part 3 of this book. The sprite editor allows you to turn on and off individual sprite pixels by using the mouse. You can design from one to four sprites of different colors with the editor. When you save the sprite object to disk, it's saved in C source code. Then you can compile the C source code that describes the sprite object(s) and use it in your program. That's the simple way. The more difficult way involves hand-coding the sprite object in C or assembly. The assembly method is easier if you use the .RADIX operator and set it to 2.

I can tell you, however, that once I had a prototype of the sprite editor up and running, I never hand-coded a sprite object again.

Summary

Bit planes are areas of memory that are mapped to physical memory starting at segment 0xA000. The more bit planes you have, the more color options your display will have. The routines in this book have been specifically designed to work with the 640 × 480, 16-color, VGA video mode.

Bit-plane animation refers to the smooth movement of graphic objects (sprites, missiles, and bullets) on one video plane. The animation functions presented in this book have been created for speed rather than other considerations. This choice means that there will be the trade-off of a potentially minor object flutter when graphic objects running on the same plane overlap.

The companion disk contains a sprite editor to facilitate the development of your sprites. The sprite objects are saved to disk as C

source files, which you can then conveniently compile and place in a library.

Chapter 2 will contain heavily documented bit-plane sample programs, and you'll learn how to have your computer move one sprite vertically, horizontally, and diagonally.

2

Moving a sprite object in a playfield space

CHAPTER 2

THIS chapter provides basic information to greatly facilitate your use of the sprite object methods in this book. The sprite object instantiation is outlined along with the presentation of sprite object methods. One sprite object animation demonstration program is contained in this chapter.

A demonstration program compiled with Borland's C/C++ version 3.1 and 4.0 compilers will show you how to move one sprite object, then you'll be formally presented with basic sprite object methods. You'll learn how to integrate the sprite object into your source file, expand the sprite object to fit all horizontal pixel positions, initialize your screen's playfield object, and finally destroy the playfield object.

Demonstration of a seamless interface to Borland's GI

Figure 2-1 provides the source code for PROG2-1.CPP. This program was designed to be compiled with Borland's C/C++ compiler. If you examine the file, you'll see how easy it is to integrate the sprite object management function into Borland's graphical environment.

Figure 2-2 presents the make file used to build the sprite object class library and to compile and link the executable program. You select the program you want to compile by changing the PROG= setting to the program name. For example, if you want to compile and link PROG2-1.CPP, you'd set PROG= to PROG2-1. If you want to compile and link GAME.CPP, you'd set PROG= to GAME. As simple as that.

You invoke the Borland make program simply by typing MAKE and hitting the Enter key. Figure 2-2 provides the listing of Borland's sprite class library make file.

Including the sprite object file

The sprite object editor presented in part two of this book writes a C source file containing data that describes the sprite object. The sprite object is described in a simple buffer. This buffer holds the data that describe the sprite object's image. Let's say you use the sprite object editor to define a single-color sprite object. If the name of the sprite object file is BOX.C (created by the sprite object editor in this case), then the sprite object definition buffer would be declared in the following fashion:

```
unsigned char BOX1[48*7*8]= { ... }
```

And then the data description of the sprite object would follow. If you wanted to include BOX.C in your source file, you'd do so in the following fashion:

```
#include "box.c"
```

And that's how simple it is to include a sprite object into your program source code. Figure 2-3 presents the source code listing for the box.c sprite object. Note that it has been automatically created by the sprite object editor.

Creating an instance of the sprite class

One of the nice features of C++ is that many complex operations are hidden from the applications programmer. When the sprite image data is loaded into memory, the image data must be shifted, twisted, and relocated in the image buffer. These convoluted machinations take place during the instantiation of the class. Here's how the sprite class is instantiated:

```
Sprite S1( BOX1, 1 );
```

where BOX1 is a label referencing a sprite object that has been created using the sprite editor. Short and sweet.

The first parameter, BOX1, contains the address of the image data buffer that has been created by the sprite editor (presented in Part 3 of the book). The second parameter represents the video plane—the sprite color to the program user.

Creating an instance of the playfield class

The playfield refers to the screen background on which the sprite object moves. Once again, invoking a C++ class constructor—in this case the playfield constructor—thankfully hides many operations from the applications programmer. Here's how easy it is to create an instance of the playfield class:

```
Playfield P1( 0 );
```

Moving the sprite instance on the playfield

The sprite class method that facilitates the movement of the sprite instance is called MoveImage(...). Here's an example of how the sprite class MoveImage(...) method is used:

```
S1.MoveImage( col, row );
```

The first and second parameters refer to the location of the upper left portion of the sprite image's column (x) and row (y) location on the 640 × 480 VGA video mode display.

Destroying the playfield

The process of destroying the playfield consists of freeing the memory previously allocated in the bit-plane animation header files. The playfield class method used to deallocate the memory is called Destroy(...). Here's an example of how the Playfield Class Destroy(...) method is invoked:

```
P1.Destroy();
```

These header files are included in your bit-plane animation source files (on the companion disk). They contain defines and structures used by both the sprite object editor and bit-plane animation programs. Figure 2-4 presents the source code listing for GOCLASS.H, which contains function prototypes of the bit-plane animation functions.

Figure 2-5 provides the source code listing for TPROTO.H, which contains function prototypes used by both the sprite object editor and bit-plane animation programs.

Figure 2-6 presents the source code listing for KEYBOARD.H, which contains defines for the keyboard key codes.

Figure 2-7 presents the source code listing for ASCII.H, which contains ASCII defines.

Figure 2-8 presents the source code listing for TSTRUCT.H, which contains structures and defines used by the sprite object editor.

Summary

This chapter contains the basic information for using bit-plane animation functions with either Borland's or Microsoft's graphical interfaces.

The sprite object is saved by the sprite object editor in standard C source code format. This permits you to include the sprite object by name in your C source file. Once the sprite object has been included, you must expand it. The expansion is required in order to facilitate the smooth sprite object, missile, or bullet motion sequences.

Sprites, missiles, and bullets move about the screen over what I call the playfield. You must create the playfield instance before any sprite object motion can take place. The object data stored when the playfield is instantiated is used by the sprite, missile, and bullet motion methods.

After the sprite instance has been created, you can incrementally move the sprite in any direction. If you want to relocate the sprite by more than one unit in any direction, you must erase it and then move it to the new location.

Once your program has finished executing, you should destroy the playfield class instance. This operation frees memory that was dynamically allocated during the playfield initialization process.

Chapter 3 will demonstrate how to move a multicolored sprite previously created by the sprite editor) smoothly about the screen.

Figure 2-1 *The source code for PROG2-1.CPP.*

```
//////////////////////////////////////
//
// prog2-1.cpp
//
// Demonstration of simple movement of
// one sprite, demonstrating a seamless
// interface to Borland's graphical
// interface (BGI)
//

//////////////////////////////////////
//
//  defines
//

#include <graphics.h>
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include "goclass.h"
#include "tproto.h"

//////////////////////////////////////
//
// include sprite created by the
// sprite editor

#include "BOX.C"

void main(void);
void __delay(int num);
```

```
void main()
{
int  row, col, key, cnt;
int param;
int gdriver= DETECT, gmode, errorcode;
int midx, midy, mode;
int max_x, max_y;

Keyboard K1(0);

// BGI graphics initialization routine

initgraph(&gdriver, &gmode, "");

// check to see if there is an error

errorcode= graphresult();

// if initialization error occurs, then report
// error and abort

if(errorcode != grOk) {
   printf("Graphics error: %s\n", grapherrormsg(errorcode));
   printf("Press any to return to DOS\n");
   K1.Get();
   exit(1);
   }

// initialize sprite

Sprite S1(BOX1, 1);

setbkcolor(LIGHTGRAY);

// set text style

settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);

// print message to screen in graphics text

setcolor(BLUE);

outtextxy(140, 20, "ENTER Runs Program -- F10 Quits to DOS");

// initialize graphics playfield after playfield
// is drawn

PlayField P1(0);

// initialize row and column location of sprite

row= 50;
col= 50;
```

Figure 2-1 *Continued.*

```
// display sprite image on screen

S1.MoveImage(col, row);

// wait for key press

key= K1.Get();

if(key == F10) {
   goto program_abort;
   }

////////////////////////////////////
// goto program execution label

restart_program:

////////////////////////////////////
// move sprite horizontally right

for(;;) {
   if(col > 560) {
      break;
      }

   // move the sprite image

         S1.MoveImage(col++, row);

   // slow down to human time frame

   __delay(12);
   }

////////////////////////////////////
// move sprite down

for(;;) {
   if(row > 400) {
      break;
      }

   // move the sprite image

   S1.MoveImage(col, row++);

   // slow down to human time frame

   __delay(12);
   }

////////////////////////////////////
// move sprite horizontally left
```

```
for(;;) {
   if(col < 50) {
      break;
      }

   // move the sprite image

  S1.MoveImage(col--, row);

   // slow down to human time frame

   __delay(12);
   }

////////////////////////////////////
// move sprite up

for(;;) {
   if(row < 50) {
      break;
      }

   // move the sprite image

  S1.MoveImage(col, row--);

   // slow down to human time frame

   __delay(12);
   }

////////////////////////////////////
// move sprite diagonally down
// right

for(;;) {
   if(row > 400) {
      break;
      }

   // move the sprite image

  S1.MoveImage(col++, row++);

   // slow down to human time frame

   __delay(12);
   }

////////////////////////////////////
// move sprite diagonally up
// left
```

Figure 2-1 *Continued.*

```
   for(;;) {
      if(row < 50) {
         break;
         }

      // move the sprite image

         S1.MoveImage(col--, row--);

         // slow down to human time frame

         __delay(12);
         }

      key= K1.Get();

      if(key != F10) {
         goto restart_program;
         }

      // program has been aborted before it is run

program_abort:

   // return from graphics mode

      closegraph();

      // free memory allocated during playfield initialization

      S1.Free();
      P1.Destroy();
}

//
////////////////////////////////////////
//
// microprocessor-sensitive delay
//

void __delay(int num)
{
int cnt1, cnt2;
int ctr1, ctr2;

   cnt1= num;
   cnt2= num;
   cnt1<<= 4;
   cnt2<<= 2;
   for(ctr1= 0; ctr1 < cnt1; ctr1++) {
      cnt1= cnt1;
      for(ctr2= 0; ctr2 < cnt2; ctr2++) {
         cnt2= cnt2;
```

```
        }
    }
}
```

The source code for Borland's sprite class library make file. Figure 2-2

```
FLAGS= -w3 -ml -v -N

OC0= sprite_a.obj sprite.obj video.obj sound.obj
OC1= key.obj playf.obj missile.obj go.obj bullet.obj
SRC= go.cpp goclass.cpp sprite.cpp key.cpp playf.cpp missile.cpp video.cpp
ALL_O= $(OC0) $(OC1)

LIBS= $(ALL_O)
PROG= prog8-1

$(PROG).exe: $(PROG).obj $(ALL_O)
        bcc $(FLAGS) $(PROG).obj graphics.lib bitplane.lib

$(PROG).obj: $(PROG).cpp
        bcc -c  $(FLAGS) $(PROG).cpp

go.obj : $(SRC)
        bcc -c $(FLAGS) go.cpp
        tlib bitplane -+go

bullet.obj : $(SRC)
        bcc -c $(FLAGS) bullet.cpp
        tlib bitplane -+bullet

sprite.obj : $(SRC)
        bcc -c $(FLAGS) sprite.cpp
        tlib bitplane -+sprite

sound.obj : $(SRC)
        bcc -c $(FLAGS) sound.cpp
        tlib bitplane -+sound

key.obj : $(SRC)
        bcc -c $(FLAGS) key.cpp
        tlib bitplane -+key

playf.obj : $(SRC)
        bcc -c $(FLAGS) playf.cpp
        tlib bitplane -+playf

missile.obj : $(SRC)
        bcc -c $(FLAGS) missile.cpp
        tlib bitplane -+missile

video.obj : $(SRC)
        bcc -c $(FLAGS) video.cpp
        tlib bitplane -+video
```

Figure 2-2 *Continued.*

```
sprite_a.obj: sprite_a.asm
        tasm /mx /zi /Dmdl=3 sprite_a.asm
        tlib bitplane -+sprite_a
```

Figure 2-3 *The source code for BOX.C.*

```
////////////////////////////////////////
// BOX.c
//

unsigned char BOX1[48*7*8]= {
   0, 0, 0, 0, 0, 0, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   96, 0, 0, 0, 0, 3, 0,
   96, 0, 0, 0, 0, 3, 0,
   96, 0, 0, 0, 0, 3
```

```
96, 0, 0, 0, 0, 3, 0,
96, 0, 0, 0, 0, 3, 0,
96, 0, 0, 0, 0, 3, 0,
96, 0, 0, 0, 0, 3, 0,
96, 0, 0, 0, 0, 3, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0
```

Figure 2-3 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 2-3 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 2-3 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
```

Figure 2-4 *Continued.*

```
  #define VOID void
#endif

#ifndef INT
  #define INT int
#endif

#ifndef GOCLASSES

////////////////////////////////////////
//
// Structure definitions
//

struct spriteInfo {
  int x;
  int y;
  int visible;
  };

struct missileInfo {
  int x;
  int y;
  int visible;
  };

#ifndef SISP
   #define SISP spriteInfo *
#endif

#ifndef MISP
   #define MISP missileInfo *
#endif

////////////////////////////////////////
//
// Class heirarchy
//

class Video {
  UCHAR current_mode;
public:
  VOID SaveMode();
  VOID RestoreMode();
  VOID SetMode(int v_mode);
  VOID SetPlane(UCHAR plane);
};

class PlayField : public Video {
public:
  UCHAR *plane1;
  UCHAR *plane2;
```

```
  UCHAR *plane3;
  UCHAR *plane4;
  int   Number;
public:
      PlayField(int i);
  void Clear();
  void Destroy();
};

class Sprite : public Video {
   int visible;
   int first_move;              // set to 0 during initialization
   UCHAR *screen_data;
   UCHAR *sprite_data;
   int current_x;               // current x location
   int current_y;               // current y location
   int old_x;                   // current x location
   int old_y;                   // current y location
   UCHAR plane;         // plane of sprite
   UCHAR *data_table; // pointer to sprite shape table
public:
      Sprite(UCHAR *dt, UCHAR plane);
  VOID SelectImage(int col);
  VOID ExpandImage(UCHAR *image_name);
  VOID EraseImage(VOID);
  VOID MoveImage(int x, int y);
  SISP GetInfo(SISP p);
  INT  SpriteCollision(spriteInfo* sip);
  INT  MissileCollision(missileInfo* mip);
  VOID RelocateImage(int x, int y);
  VOID RestoreScreenImage(VOID);
  VOID SaveScreenImage(int x, int y);
  VOID Free(VOID);
  VOID shift_sprite(VOID);              // internal management function
  VOID erasprite(int x, int y);    // internal management function
  VOID mvsprite(int x, int y);     // internal management function
};

class Missile : public Video {
   int visible;
   int first_move;              // set to 0 during initialization
   UCHAR *screen_data;
   UCHAR *m_data;
   int current_x;               // current x location
   int current_y;               // current y location
   int old_x;                   // current x location
   int old_y;                   // current y location
   UCHAR plane;         // plane of sprite
   UCHAR *data_table; // pointer to missile shape table
public:
      Missile(UCHAR s_plane);
   VOID SelectImage(int col);
   VOID MoveImage(int x, int y);
```

Figure 2-4 *Continued.*

```
    VOID EraseImage();
    MISP GetInfo(MISP p);
    VOID Free();
    VOID mvmissile(int x, int y);

};

class Bullet : public Video {
    int visible;
    int first_move;              // set to 0 during initialization
    UCHAR *screen_data;
    UCHAR *b_data;
    int current_x;               // current x location
    int current_y;               // current y location
    int old_x;                   // current x location
    int old_y;                   // current y location
    UCHAR plane;        // plane of sprite
    UCHAR *data_table; // pointer to missile shape table
public:
         Bullet(UCHAR s_plane);
    VOID SelectImage(int col);
    VOID MoveImage(int x, int y);
    VOID EraseImage();
    VOID Free();
    VOID mvbullet(int x, int y);

};

class Keyboard {
  int lastKey;
public:
       Keyboard(int i);
  int Get();
  int Status();
  int LastKey() { return lastKey; }
};

class Sound {
   int tone;
public:
     Sound::Sound(int freq);
VOID Sound::Restore();
VOID Sound::On(int freq);
VOID Sound::Off();
};

#endif

#ifndef GOCLASSES
   #define GOCLASSES
#endif
```

The source code for TPROTO.H. Figure 2-5

```
//////////////////////////////////////////
//
// tproto.h
//
// function prototypes for
// game I/O library
//
//////////////////////////////////////////

//////////////////////////////////////////
//
// include header files here
//

#include "keyboard.h"
#include "ascii.h"
#include "tstruct.h"

#ifndef MK_FP
#define MK_FP(seg,ofs)  ((void far *) \
                        (((unsigned long)(seg) << 16) | (unsigned)(ofs)))
#endif

void mson(void);
void msoff(void);
int  msinit(void);
int  msstat(int *x, int *y);

//////////////////////////////////////////
// string routines

void delChar(char *);
int  findChar(int, char *);
void insChar(char *, char);
void insNum(char *,int);
char *memichr(char *,char,int);
char *ritoa(int,char *,int,int,char);
void strCjust(char *);
char *strins(char *,char *,int);
void stripblk(char *);
void strEnul(char *);
void strLjust(char *);
void strRjust(char *);
int  strAnal(int *,int *,char *);

//////////////////////////////////////////
// keyboard routines

char gtChar(void);
int  gtKey(void);
char gtScan(void);
int  gtKBstat(void);
```

Figure 2-5 *Continued.*

```
int  gtKBflag(void);
int  gtKBflsh(int);

////////////////////////////////////////
// cursor routines

void gtCur(int *, int *);
int  g_shape(void);
void mvCur(int, int);
void offCur(void);
void onCur(void);
void ssizeCur(void);
void rsizeCur(void);
void sizeCur(int, int);
void sCloc(void);
void s_shape(int);

////////////////////////////////////////
// rectangle routines

RECT *setRect(RECT *, int, int, int, int);
void addRect(RECT *, RECT *);
void subRect(RECT *, RECT *);
void dupRect(RECT *, RECT *);
void dsyRect(RECT *);
void offRect(RECT *, int, int );
void boxRect(RECT *, int, int);
void clrRect(RECT *);
void fillRect(RECT *, int);
void saveRect(RECT *);
unsigned int sizeRect(RECT *);
void restRect(RECT *);
void scUp(RECT *, int, int);
void scDn(RECT *, int, int);

////////////////////////////////////////
// sound routines

void beep(void);
void bleep(void);
void offSound(void);
void onSound(int);

////////////////////////////////////////
// print routines

int  prChar(int,char);
int  prInit(int);
int  prScrn(int);
int  prScrnFF(int);
int  prStatus(int);
```

```
////////////////////////////////////////
// screen routines

void ascup(int,int,int,int,int,int,int);
void scrnClr(void);
void vdAttr(int, int, int, int);
void vdChar(int, int, int);
void vdChr(int, int, int);
void vdHoriz(int, int, int, int);
void vdStr(int, int, int, char *, char);
void vdWrite(int, int, int, char *, int);
void vidInit(void);
void saveScrn(void);
void restScrn(void);

////////////////////////////////////////
// window routines

void clrWind(WIND *);
void dispWind(WIND *);
void dsyWind(WIND *);
void rdImg(WIND *);
void rdWind(WIND *);
void remvWind(WIND *);
void scrnClr(void);
void setAttr(WIND *, int);
void setBord(WIND *, int);
void setTitle(WIND *,char *);
WIND *setWind(WIND *, int,int,int,int);
void strtWind(WIND *);
unsigned int sizeImg(WIND *);
void wmvCur(WIND *, int, int);
void wrBox(WIND *);
void wrImg(WIND *);
void wrWind(WIND *);
void wvdAttr(WIND *, int, int, int,char);
void wvdHoriz(WIND *,int,int,int,int);
int  wvdprmne(WIND *, char *, int, int, int, int);
int  wvdprmpt(WIND *, char *, int, int, int, int);
void wvdVert(WIND *, int,int,int,int);
void wvdScdn(WIND *, int);
void wvdScup(WIND *, int);
void wvdStr(WIND *, int, int, int, char *, char);
void wvdWrite(WIND *, int, int, int, char *, char);
int  wvrdChar(WIND *, int, int);

////////////////////////////////////////
// misc. routines

void Delay(int, int);
int  mkToken(int,int);
int  mkAttr(int, int, int, int);
void exit_bad(char *);
```

Figure 2-6 *The source code for KEYBOARD.H.*

```
////////////////////////////////////////
//
// keyboard.h
//
// keyboard scan and char codes
//
////////////////////////////////////////

#define INSERT      0x5200
#define DELETE      0x5300
#define SPACE       0x3920
#define ESC         0x011b
#define ESCAPE      0x011b
#define PGDN        0x5100
#define PGUP        0x4900
#define PERIOD      0x342e
#define TAB         0x0f09
#define RT_SQUARE   0x1b5d
#define LT_SQUARE   0x1a5b
#define RT_BRACKET  0x1b7d
#define LT_BRACKET  0x1a7b
#define CNTL_HOME   0x7700
#define CNTL_END    0x7500
#define CNTL_ENTER  0x1c0a
#define CNTL_BS     0x0e7f
#define HOME        0x4700
#define END         0x4f00
#define s_BS        0x0008
#define BS          0x0e08
#define BACKSPACE   0x0e08
#define s_CR        0x000d
#define CR          0x1c0d
#define ENTER       0x1c0d
#define UP_ARROW    0x4800
#define RIGHT_ARROW 0x4d00
#define LEFT_ARROW  0x4b00
#define DOWN_ARROW  0x5000
#define F1          0x3b00
#define F2          0x3c00
#define F3          0x3d00
#define F4          0x3e00
#define F5          0x3f00
#define F6          0x4000
#define F7          0x4100
#define F8          0x4200
#define F9          0x4300
#define F10         0x4400

#define SHIFT_TAB   0x0f00
#define SHIFT_HOME  0x4737
#define SHIFT_END   0x4f31
```

```
#define SHIFT_INSERT 0x5230
#define SHIFT_DELETE 0x532e
#define SHFT_INSERT  0x5230
#define SHFT_F1      0x5400
#define SHFT_F2      0x5500
#define SHFT_F3      0x5600
#define SHFT_F4      0x5700
#define SHFT_F5      0x5800
#define SHFT_F6      0x5900
#define SHFT_F7      0x5a00
#define SHFT_F8      0x5b00
#define SHFT_F9      0x5c00
#define SHFT_F10     0x5d00
#define SH_R_ARROW   0x4d36
#define SH_L_ARROW   0x4b34
#define SH_U_ARROW   0x4838
#define SH_D_ARROW   0x5032

#define CNTL_F1      0x5e00
#define CNTL_F2      0x5f00
#define CNTL_F3      0x6000
#define CNTL_F4      0x6100
#define CNTL_F5      0x6200
#define CNTL_F6      0x6300
#define CNTL_F7      0x6400
#define CNTL_F8      0x6500
#define CNTL_F9      0x6600
#define CNTL_F10     0x6700
#define CNTL_LEFTA   0x7300
#define CNTL_RIGHTA  0x7400

#define ALT_F1       0x6800
#define ALT_F2       0x6900
#define ALT_F3       0x6a00
#define ALT_F4       0x6b00
#define ALT_F5       0x6c00
#define ALT_F6       0x6d00
#define ALT_F7       0x6e00
#define ALT_F8       0x6f00
#define ALT_F9       0x7000
#define ALT_F10      0x7100

#define ALT_A        0x1e00
#define ALT_B        0x3000
#define ALT_C        0x2e00
#define ALT_D        0x2000
#define ALT_E        0x1200
#define ALT_F        0x2100
#define ALT_G        0x2200
#define ALT_H        0x2300
#define ALT_I        0x1700
#define ALT_J        0x2400
#define ALT_K        0x2500
```

Figure 2-6 *Continued.*

```
#define ALT_L          0x2600
#define ALT_M          0x3200
#define ALT_N          0x3100
#define ALT_O          0x1800
#define ALT_P          0x1900
#define ALT_Q          0x1000
#define ALT_R          0x1300
#define ALT_S          0x1f00
#define ALT_T          0x1400
#define ALT_U          0x1600
#define ALT_V          0x2f00
#define ALT_W          0x1100
#define ALT_X          0x2d00
#define ALT_Y          0x1500
#define ALT_Z          0x2c00

#define CNTL_A         0x1e01
#define CNTL_B         0x3002
#define CNTL_C         0x2e03
#define CNTL_D         0x2004
#define CNTL_E         0x1205
#define CNTL_F         0x2106
#define CNTL_G         0x2207
#define CNTL_H         0x2308
#define CNTL_I         0x1709
#define CNTL_J         0x240a
#define CNTL_K         0x250b
#define CNTL_L         0x260c
#define CNTL_M         0x320d
#define CNTL_N         0x310e
#define CNTL_O         0x180f
#define CNTL_P         0x1910
#define CNTL_Q         0x1011
#define CNTL_R         0x1312
#define CNTL_S         0x1f13
#define CNTL_T         0x1414
#define CNTL_U         0x1615
#define CNTL_V         0x2f16
#define CNTL_W         0x1117
#define CNTL_X         0x2d18
#define CNTL_Y         0x1519
#define CNTL_Z         0x2c1a

#define K_0            0x0b30
#define K_1            0x0231
#define K_2            0x0332
#define K_3            0x0433
#define K_4            0x0534
#define K_5            0x0635
#define K_6            0x0736
#define K_7            0x0837
#define K_8            0x0938
#define K_9            0x0a39
```

```
#define ALT_0          0x8100
#define ALT_1          0x7800
#define ALT_2          0x7900
#define ALT_3          0x7a00
#define ALT_4          0x7b00
#define ALT_5          0x7c00
#define ALT_6          0x7d00
#define ALT_7          0x7e00
#define ALT_8          0x7f00
#define ALT_9          0x8000

#define K_SPACE        0x3920
#define K_EXCLAM       0x0221
#define K_QUOTE        0x2822
#define K_POUND        0x0423
#define K_DOLLAR       0x0524
#define K_PERCENT      0x0625
#define K_AND          0x0826
#define K_APOST        0x2827
#define K_LPAREN       0x0A28
#define K_RPAREN       0x0B29
#define K_STAR         0x092A
#define K_PLUS         0x0D2B
#define K_COMMA        0x332C
#define K_MINUS        0x0C2D
#define K_PERIOD       0x342E
#define K_FSLASH       0x352F

#define K_COLON        0x273A
#define K_SCOLON       0x273B
#define K_LESS         0x333C
#define K_EQUAL        0x0D3D
#define K_GREAT        0x343E
#define K_QUEST        0x353F
#define K_AMPER        0x0340

#define K_A            0x1E61 - 0x20
#define K_B            0x3062 - 0x20
#define K_C            0x2E63 - 0x20
#define K_D            0x2064 - 0x20
#define K_E            0x1265 - 0x20
#define K_F            0x2166 - 0x20
#define K_G            0x2267 - 0x20
#define K_H            0x2368 - 0x20
#define K_I            0x1769 - 0x20
#define K_J            0x246A - 0x20
#define K_K            0x256B - 0x20
#define K_L            0x266C - 0x20
#define K_M            0x326D - 0x20
#define K_N            0x316E - 0x20
#define K_O            0x186F - 0x20
#define K_P            0x1970 - 0x20
#define K_Q            0x1071 - 0x20
```

Figure 2-6 *Continued.*

```
#define K_R             0x1372 - 0x20
#define K_S             0x1F73 - 0x20
#define K_T             0x1474 - 0x20
#define K_U             0x1675 - 0x20
#define K_V             0x2F76 - 0x20
#define K_W             0x1177 - 0x20
#define K_X             0x2D78 - 0x20
#define K_Y             0x1579 - 0x20
#define K_Z             0x2C7A - 0x20

#define K_LBRACK        0x1A5B
#define K_BSLASH        0x2B5C
#define K_RBRACK        0x1B5D
#define K_KARAT         0x075E
#define K_UNDER         0x0C5C

#define K_a             0x1E61
#define K_b             0x3062
#define K_c             0x2E63
#define K_d             0x2064
#define K_e             0x1265
#define K_f             0x2166
#define K_g             0x2267
#define K_h             0x2368
#define K_i             0x1769
#define K_j             0x246A
#define K_k             0x256B
#define K_l             0x266C
#define K_m             0x326D
#define K_n             0x316E
#define K_o             0x186F
#define K_p             0x1970
#define K_q             0x1071
#define K_r             0x1372
#define K_s             0x1F73
#define K_t             0x1474
#define K_u             0x1675
#define K_v             0x2F76
#define K_w             0x1177
#define K_x             0x2D78
#define K_y             0x1579
#define K_z             0x2C7A
```

Figure 2-7 *The source code for ASCII.H.*

```
//////////////////////////////////////////
//
// ascii.h
//
// ASCII def header file
//
//////////////////////////////////////////
```

```
#define   aNUL      0     //      null \0 delimeter
#define   aSOH      1     // ^A - start of heading
#define   aSTX      2     // ^B - start of text
#define   aETX      3     // ^C - end of text
#define   aEOT      4     // ^D - end of transmission
#define   aENQ      5     // ^E - inquiry
#define   aACK      6     // ^F - affirm acknowledgement
#define   aBEL      7     // ^G - audible bell
#define   aBS       8     // ^H - backspace
#define   aTAB      9     // ^I - horizontal tab
#define   aLF       10    // ^J - line feed
#define   aVT       11    // ^K - vertical tab
#define   aFF       12    // ^L - form feed
#define   aCR       13    // ^M - carriage return
#define   aSO       14    // ^N - shift out
#define   aSI       15    // ^O - shift in
#define   aDCE      16    // ^P - data link escape
#define   aDC1      17    // ^Q - device control 1
#define   aDC2      18    // ^R - device control 2
#define   aDC3      19    // ^S - device control 3
#define   aDC4      20    // ^T - device control 4
#define   aNAK      21    // ^U - neg acknowledge
#define   aSYN      22    // ^V - synchronous idle
#define   aETB      23    // ^W - end of transmission
#define   aCAN      24    // ^X - cancel
#define   aEM       25    // ^Y - end of medium
#define   aSUB      26    // ^Z - substitute
#define   aESC      27    //      escape
#define   aFS       28    //      file separator
#define   aGS       29    //      group separator
#define   aRS       30    //      record separator
#define   aUS       31    //      unlinked separator
#define   aSPC      32    //      space

#define   aCODE     94    // ^ then printer commands follow
#define   aHCR      aEOT  // Hard carriage return
#define   aCENTER   'C'   // code to center line
#define   aDOUBLE   'D'   // double strike toggle
#define   aEXPAND   'E'   // emphasize toggle
#define   aSUPERS   'S'   // superscript toggle
#define   aITALIC   'I'   // italics toggle
#define   aBOLD     'B'   // bold toggle

#define   aTRUE     1     // true
#define   aFALSE    0     // false
```

The source code for TSTRUCT.H.

Figure 2-8

```
////////////////////////////////////////
//
// tstruct.h
//
//
```

Figure 2-8 *Continued.*

```
#define IMAGE unsigned int

////////////////////////////////////////
// defines

#define LOTUS_ITEM_MAX 20
#define MENUBAR_ITEM_MAX 10

////////////////////////////////////////
//
// structure for lotus-style window interface
//

typedef struct  {
   int  number;                          // number of LOTUS objects
   char *name[LOTUS_ITEM_MAX];           // pointer to item name
   char *explain[LOTUS_ITEM_MAX];        // pointer to item explanation
   int  lot_map[LOTUS_ITEM_MAX][2];      // map for lotus item highlights
   int  lotus_item;                      // highlight and
   int  old_lotus;                       // item selection data
   int  lotus_open;                      // status of lotus window
   unsigned int imgbuf[160];             // top two rows of screen image
   } LOTUS_CLASS;

////////////////////////////////////////
//
// structure for lotus-style window interface
//

typedef struct  {
   int  number;                          // number of MENUBAR objects
   char *name[MENUBAR_ITEM_MAX];         // pointer to item name
   int  mb_map[MENUBAR_ITEM_MAX][2];     // map for menubar item highlights
   int  key_list[MENUBAR_ITEM_MAX];      // list of hotkeys
   int  menubar_item;                    // highlight and
   int  old_menubar;                     // item selection data
   int  menubar_open;                    // status of lotus window
   int  si_attr;                         // item attribute
   int  sinv_attr;                       // inverse attribute
   int  sk_attr;                         // highlight key attribute
   int  first_time;                      // first time
   unsigned int imgbuf[160];             // top two rows of screen image
   } MENUBAR_CLASS;

////////////////////////////////////////
// rectangle structure

typedef struct {
   int ul_row;                           // upper left row
   int ul_col;                           // upper left column
   int lr_row;                           // lower right row
   int lr_col;                           // lower right column
```

```
   unsigned int *image;                  // pointer to scrn image
} RECT;

////////////////////////////////////////
// cursor structure

typedef struct {
   int row;                              // cursor row
   int column;                           // cursor column
} CUR_LOCATION;

////////////////////////////////////////
//************************************
// DOS structures
//

#ifndef OS2_PROG

//
////////////////////////////////////////

////////////////////////////////////////
// window structure

typedef struct {
int ul_row;                              // upper left row
int ul_col;                              // upper left column
int lr_row;                              // lower right row
int lr_col;                              // lower right column
unsigned int img_size;                   // window img size
unsigned int far *img_ptr;               // pointer scrn image
unsigned int far *wind_ptr;              // pointer scrn image
int box_type;                            // border selection
int attr;                                // window attribute
int visible;                             // window on
int top_offset;                          // col offset title
int top_length;                          // length title str
int show_top;                            // display title
int bot_offset;                          // col offset title
int bot_length;                          // length title str
int show_bot;                            // display title
char *t_title;                           // ptr to t title str
   char *b_title;                        // ptr to b title str
} WIND;

////////////////////////////////////////
// video structure

typedef struct {
   int mode;                             // video mode
   int row_width;                        // columns per row
   int page;                             // video page
   unsigned int far *scrn;               // pointer to video RAM
} VIDEO;
```

Figure 2-8 *Continued.*

```
//////////////////////////////////////////
//
// otherwise OS2 structures

#else

//
//////////////////////////////////////////

typedef struct {
   int ul_row;                          // upper left row
   int ul_col;                          // upper left column
   int lr_row;                          // lower right row
   int lr_col;                          // lower right column
   unsigned int img_size;               // window img size
   unsigned int  *img_ptr;              // pointer scrn image
   unsigned int  *wind_ptr;             // pointer scrn image
   int box_type;                        // border selection
   int attr;                            // window attribute
   int visible;                         // window on
   int top_offset;                      // col offset title
   int top_length;                      // length title str
   int show_top;                        // display title
   int bot_offset;                      // col offset title
   int bot_length;                      // length title str
   int show_bot;                        // display title
   char *t_title;                       // ptr to t title str
   char *b_title;                       // ptr to b title str
} WIND;

typedef struct {
   int mode;                            // video mode
   int row_width;                       // columns per row
   int page;                            // video page
   unsigned int  *scrn;                 // pointer to video RAM
} VIDEO;

//////////////////////////////////////////
//
// end of platform-dependent conditions

#endif

//
//*****************************************
//////////////////////////////////////////

//////////////////////////////////////////
// misc. defines
```

```
#define S_S_S_S             0
#define S_S_D_D             1
#define D_D_S_S             2
#define D_D_D_D             3
#define BLACK               0
#define BLUE                1
#define GREEN               2
#define CYAN                3
#define RED                 4
#define MAGENTA             5
#define BROWN               6
#define WHITE               7
#define NORMAL              7
#define REVERSE             112
#define ON_INTENSITY        8
#define OFF_INTENSITY       0
#define ON_BLINK            128
#define OFF_BLINK           0
#define UP_SCROLL           6
#define DOWN_SCROLL         7
#define PRINT_TIME_OUT      1
#define IO_ERROR            4
#define PRINT_SELECTED      8
#define OUT_OF_PAPER        16
#define ACKNOWLEDGE         32
#define PRINT_NOT_BUSY      64
#define ON_ECHO_CTRL        1
#define OFF_ECHO_CTRL_C     7
#define OFF_ECHO            8
#define RIGHT_SHIFT         1
#define LEFT_SHIFT          2
#define CLRL_PRESS          4
#define ALT_PRESS           8
#define SCROLL_LOCK         16
#define NUM_LOCK            32
#define CAPS_LOCK           64
#define INSERT_ON           128
#define CENTER              0xff
#define NUMBERED            1
#define RESETROW            2
#define UPPER               1
#define LOWER               2
#define NAME                3
#define LEFTBUTTON          1
#define RIGHTBUTTON         2
#define CNTRBUTTON          4
```

3

Moving a three-colored sprite

CHAPTER 3

IN this chapter a demonstration program will show you how to display and move a three-colored sprite. As you'll see, it's nearly as easy as moving a one-colored sprite. There is, however, one serendipitous condition that can provide a slight bit of pizazz to your multicolored sprite animation display.

When using the sprite management functions presented in this book, think of multicolored sprites as running on different planes that overlap. The color of the sprite is determined by the plane on which the sprite moves, so the algorithm for moving multicolored or overlapping sprites is:

❶ Move sprite 1.

❷ Move sprite 2.

❸ Move sprite 3.

❹ Delay.

When you move the three different sprites, the multicolored sprite moves. When you run PROG3-1.EXE you'll notice that when the three-colored sprite moves in a horizontal direction, a minuscule white space appears between the leading blue edge and the red bar in the center of the sprite. Why do you think that happens?

There are times when the display of the first sprite and the second sprite are a pixel out of synch. You can use this condition to your advantage if you order the movement of the sprites in a conscious fashion. This effect might be considered a serendipitous attempt at within-sprite animation. Within-sprite animation refers to the shape of the graphic object itself changing, as opposed to the graphic object changing location on the playfield.

For example, the within-sprite sample animation program PROG3-1.CPP uses conscious selection of an individual sprite within the multicolored sprite to have its eyes look in the direction of its motion. The effect is subtle and effective.

Let's take a detailed look at how to move a multicolored sprite on the graphics playfield.

Moving a multicolored sprite on the graphics playfield

Figure 3-1 presents the source code of PROG3-1.CPP. This program demonstrates the movement of a multicolored sprite using the sprite management functions presented in the book. Figure 3-2 presents the source code for BOXA.C, which contains the descriptive data for the multicolored sprite used in prog3-1.exe (Fig. 3-1).

The procedure for moving a multicolored sprite is very similar to that for moving a single-colored sprite. If you look at the source code in PROG3-1.CPP, you'll notice there's a source line:

```
#include "BOXA.C"
```

near the top of the file. The BOXA.C file was created using the sprite editor presented in Part 3 of this book. The source code for all three sprites is contained in BOXA.C. The names of the sprites are BOXA1, BOXA2, and BOXA3.

Once the sprite data for the three sprites composing the multicolored sprite is included, you need to create instances of the sprite class. Here's how it's done:

```
Sprite S1 (BOXA1, 1);
Sprite S2 (BOXA2, 2);
Sprite S3 (BOXA3, 3);
```

Do you see how simple it is to create an instance for the three sprite images that make up the multicolored sprite? Next, create an instance of the playfield class:

```
Playfield P1( 0 );
```

Now you're ready to display the multicolored sprite:

```
S3.MoveImage(col, row);
S2.MoveImage(col, row);
S1.MoveImage(col, row);
```

There's no magic here. It's just as easy to display a single-color sprite image as it is to display a multicolored sprite image designed by the sprite editor.

Summary

When using the sprite management functions presented in this book, a multicolored sprite is no more than separate overlapping sprites moving on different planes. The sprite editor facilitates the creation of multicolored (multiplane) sprites.

Under certain conditions, the first sprite that's moved in a multicolored sprite will visually appear to lead the other sprites. You can use this quirk to your advantage when displaying multicolored graphic objects.

Chapter 4 will introduce a technique to animate the sprite image change the sprite image itself) while it's moving about the screen.

Figure 3-1 *The source code for PROG3-1.CPP.*

```
//////////////////////////////////////////
//
// prog3-1.cpp
//
// Demonstration of simple movement of
// one sprite
//

//////////////////////////////////////////
//
//  defines
//

#include <graphics.h>
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include "tproto.h"

#include "goclass.h"

//////////////////////////////////////////
//
// include sprite created by the
// sprite editor
```

```
#include "BOXA.C"

void main(void);

void main()
{
int  row, col, key, cnt;
int param;
int gdriver= DETECT, gmode, errorcode;
int midx, midy, mode;
int max_x, max_y;

  // initialize Keyboard class constructor

  Keyboard K1(0);

   // BGI graphics initialization routine

   initgraph(&gdriver, &gmode, "");

   // check to see if there is an error

   errorcode= graphresult();

   // if initialization error occurs, then report
   // error and abort

   if(errorcode != grOk) {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any to return to DOS\n");
      K1.Get();
      exit(1);
      }

   // initialize the sprite

   Sprite S1(BOXA1, 1);
   Sprite S2(BOXA2, 2);
   Sprite S3(BOXA3, 4);

   setbkcolor(LIGHTGRAY);

   // set text style

   settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);

   // print message to screen in graphics text

   setcolor(BLUE);

   outtextxy(140, 20, "ENTER Runs Program -- F10 Quits to DOS");

   // initialize graphics playfield after playfield
```

Figure 3-1 *Continued.*

```
// is drawn

PlayField P1(0);

// initialize row and column location of sprite

row= 50;
col= 50;

// display sprite image on screen

S3.MoveImage(col, row);
S2.MoveImage(col, row);
S1.MoveImage(col, row);

// wait for key press

key= K1.Get();

if(key == F10) {
   goto program_abort;
   }

   ////////////////////////////////////
   // goto program execution label

restart_program:

      setbkcolor(LIGHTGRAY);
      setcolor(LIGHTGRAY);

      delay(2);

      line(140, 10, 140 + (40 * 8), 10);
      for(cnt= 10; cnt < 30; cnt++) {
         line(140, cnt, 140 + (40 * 8), cnt);
         }
      delay(2);

   ////////////////////////////////////
// move sprite horizontally right

for(;;) {
   if(col > 560) {
      break;
      }

   // move the sprite image

     S3.MoveImage(col++, row);
     S2.MoveImage(col, row);
     S1.MoveImage(col, row);
```

```
   // slow down to human time frame

   delay(8);
   }

////////////////////////////////////
// move sprite down

for(;;) {
   if(row > 400) {
      break;
      }

   // move the sprite image

     S3.MoveImage(col, row++);
     S2.MoveImage(col, row);
     S1.MoveImage(col, row);

   // slow down to human time frame

   delay(8);
   }

////////////////////////////////////
// move sprite horizontallty left

for(;;) {
   if(col < 50) {
      break;
      }

   // move the sprite image

     S3.MoveImage(col--, row);
     S2.MoveImage(col, row);
     S1.MoveImage(col, row);

   // slow down to human time frame

   delay(8);
   }

////////////////////////////////////
// move sprite up

for(;;) {
   if(row < 50) {
      break;
      }

   // move the sprite image
```

Figure 3-1 *Continued.*

```
            S3.MoveImage(col, row--);
            S2.MoveImage(col, row);
            S1.MoveImage(col, row);

         // slow down to human time frame

         delay(8);
         }

         ////////////////////////////////////
         // move sprite diagonally down
         // right

         for(;;) {
            if(row > 400) {
               break;
               }

            // move the sprite image

              S3.MoveImage(col++, row++);
              S2.MoveImage(col, row);
              S1.MoveImage(col, row);

            // slow down to human time frame

            delay(8);
            }

         ////////////////////////////////////
         // move sprite diagonally up
         // left

         for(;;) {
            if(row < 50) {
               break;
               }

            // move the sprite image

              S3.MoveImage(col--, row--);
              S2.MoveImage(col, row);
              S1.MoveImage(col, row);

            // slow down to human time frame

            delay(8);
            }

         setcolor(BLUE);

         outtextxy(140, 20, "E");
         outtextxy(140, 20, "ENTER Runs Program -- F10 Quits to DOS");
```

```
    key= K1.Get();

    if(key != F10) {
       goto restart_program;
       }

    // program has been aborted before it is run

program_abort:

    // return from graphics mode

    closegraph();

    // free memory allocated during playfield initialization
  S1.Free();
  S2.Free();
  S3.Free();
  P1.Destroy();
}
```

The source code for BOXA.C.

Figure 3-2

```
// ////////////////////////////////////
// boxa.c
//

unsigned char BOXA1[48*7*8]= {
   0, 0, 0, 0, 0, 0, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   96, 0, 0, 0, 0, 3, 0,
   96, 0, 0, 0, 0, 3, 0,
   96, 0, 0, 0, 0, 3, 0,
   96, 0, 0, 0, 0, 3, 0,
   96, 0, 0, 0, 0, 3, 0,
   96, 0, 0, 0, 0, 3, 0,
   96, 0, 0, 0, 0, 3, 0,
   96, 0, 0, 0, 0, 3, 0,
   96, 0, 0, 0, 0, 3, 0,
   96, 0, 0, 0, 0, 3, 0,
   96, 0, 0, 0, 0, 3, 0,
   96, 0, 0, 0, 0, 3, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
```

Figure 3-2 *Continued.*

```
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
96, 0, 0, 0, 0, 3, 0,
96, 0, 0, 0, 0, 3, 0,
96, 0, 0, 0, 0, 3, 0,
96, 0, 0, 0, 0, 3, 0,
96, 0, 0, 0, 0, 3, 0,
96, 0, 0, 0, 0, 3, 0,
96, 0, 0, 0, 0, 3, 0,
96, 0, 0, 0, 0, 3, 0,
96, 0, 0, 0, 0, 3, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 3-2 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 3-2 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 3-2 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
31, 255, 255, 255, 255, 252, 0,
31, 255, 255, 255, 255, 252, 0,
31, 255, 255, 255, 255, 252, 0,
31, 255, 255, 255, 255, 252, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
31, 255, 255, 255, 255, 252, 0,
31, 255, 255, 255, 255, 252, 0,
31, 255, 255, 255, 255, 252, 0,
31, 255, 255, 255, 255, 252, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0
```

Figure 3-2 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 3-2 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 3-2 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 3-2 *Continued.*

```
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0 };

unsigned char BOXA3[48*7*8]= {
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    31, 255, 255, 255, 255, 252, 0,
    31, 255, 255, 255, 255, 252, 0,
    31, 255, 255, 255, 255, 252, 0,
    31, 255, 255, 255, 255, 252, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    31, 255, 255, 255, 255, 252, 0,
    31, 255, 255, 255, 255, 252, 0,
    31, 255, 255, 255, 255, 252, 0,
    31, 255, 255, 255, 255, 252, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 3-2 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 3-2 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 3-2 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

4

A within-sprite animation technique

CHAPTER 4

CHAPTER 3 presented the method for displaying a multicolored sprite by overlapping different sprites on different planes. The multiple sprite images were created with the sprite editor. Chapter 4 adds a bit more sophistication to your bit-plane animation skill.

The demonstration program PROG4-1.CPP moves a ghost-like rendering in the same path as the three-colored block in PROG3-1.CPP back in chapter 3, and the red googlie eyes of the blue ghost point in the direction the ghost is moving. It also adds a new twist—the legs of the ghost sway in the opposite direction of the motion of the ghost. In other words, the ghost moves about the screen in a smoothly animated fashion. The ghost shape is changed, or animated, according to the direction in which the ghost travels. The effect is compelling.

Changing a multicolored sprite shape as direction changes

The technique is simple. First, determine the directions your sprite will move on the graphics playfield. Second, create sprite shapes that correspond to the directions in which the sprite moves. That's it.

Figure 4-1 presents the source listing of PROG4-1.CPP. This program demonstrates one method for changing a multicolored sprite shape according to the direction in which the sprite moves. PROG4-1.CPP always knows which direction the sprite is travelling. By carefully examining the source, it'll prove easy to determine what sprite shape is used for each direction.

Each demonstration program presented in this book is incrementally more complex than the previous one. If you understood the source code presented in PROG3-1.CPP (Fig. 3-1), you should have no problem understanding the source presented in PROG4-1.CPP.

Figures 4-2, 4-3, and 4-4 present the source code for, respectively, UGHOST.C, LGHOST.C, and RGHOST.C, which were created with the sprite editor shown in Part 3 of the book.

Summary

Chapter 4 presents one method of changing the sprite shape as it gallivants about the graphics playfield. This is accomplished by using the sprite editor to create a corresponding sprite image for each direction you want the sprite to move.

You can further expand this technique by adding sprite shape images. The more sprite images you have to describe the sprite, the more professional your bit-plane animation display will appear.

Chapter 5 introduces missiles, bullets, and sound.

The source code for PROG4-1.CPP. Figure 4-1

```
//////////////////////////////////////////
//
// prog4-1.cpp
//
// Two-color sprite movement with
// sprite shape-shift emphasizing
// direction
//

//////////////////////////////////////////
//
//  defines
//

#include <graphics.h>
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>

#include "tproto.h"

#include "goclass.h"

#include "lghost.c"
#include "rghost.c"
#include "ughost.c"

void main(void);
```

Figure 4-1 *Continued.*

```
// unsigned _stklen= 32760;

void main()
{
int  row, col, key, cnt;
int param;
int gdriver= DETECT, gmode, errorcode;
int midx, midy, mode;
int max_x, max_y;

  // initialize keyboard constructor

  Keyboard K1(0);

   // BGI graphics initialization routine

   initgraph(&gdriver, &gmode, "");

   // check to see if there is an error

   errorcode= graphresult();

   // if initialization error occurs, then report
   // error and abort

   if(errorcode != grOk) {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any to return to DOS\n");
      key= K1.Get();
      exit(1);
      }

   // initialize the sprite

Sprite U1(UGHOST1, 1);
Sprite U2(UGHOST2, 4);

Sprite L1(LGHOST1, 1);
Sprite L2(LGHOST2, 4);

Sprite R1(RGHOST1, 1);
Sprite R2(RGHOST2, 4);

   setbkcolor(LIGHTGRAY);

   // set text style

   settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);

   // print message to screen in graphics text

   setcolor(BLUE);
```

```
outtextxy(140, 20, "ENTER Runs Program -- F10 Quits to DOS");

// initialize graphics playfield after playfield
// is drawn

PlayField P1(0);

// initialize row and column location of sprite

 row= 50;
 col= 50;

 // display sprite image on screen

U2.MoveImage(col, row);
U1.MoveImage(col, row);

 // wait for key press

 key= K1.Get();

 if(key == F10) {
    goto program_abort;
    }

    ////////////////////////////////////
    // goto program execution label

restart_program:

       setbkcolor(LIGHTGRAY);
       setcolor(LIGHTGRAY);

       delay(2);

       line(140, 10, 140 + (40 * 8), 10);
       for(cnt= 10; cnt < 30; cnt++) {
          line(140, cnt, 140 + (40 * 8), cnt);
          }
       delay(2);

    ////////////////////////////////////
// move sprite horizontally right

for(;;) {
   if(col > 560) {
      break;
      }

   // move the sprite image

  R1.MoveImage(col++, row);
  R2.MoveImage(col, row);
```

Figure 4-1 *Continued.*

```
        // slow down to human time frame

        delay(8);
        }

    ////////////////////////////////////
    // move sprite down

    for(;;) {
       if(row > 400) {
          break;
          }

       // move the sprite image

      U1.MoveImage(col, row++);
      U2.MoveImage(col, row);

       // slow down to human time frame

       delay(8);
       }

    ////////////////////////////////////
    // move sprite horizontallty left

    for(;;) {
       if(col < 50) {
          break;
          }

       // move the sprite image

      L1.MoveImage(col--, row);
      L2.MoveImage(col, row);

       // slow down to human time frame

       delay(8);
       }

    ////////////////////////////////////
    // move sprite up

    for(;;) {
       if(row < 50) {
          break;
          }

       // move the sprite image

      U1.MoveImage(col, row--);
      U2.MoveImage(col, row);

       // slow down to human time frame
```

```
   delay(8);
   }

////////////////////////////////////
// move sprite diagonally down
// right

for(;;) {
   if(row > 400) {
      break;
      }

   // move the sprite image

  R1.MoveImage(col++, row++);
  R2.MoveImage(col, row);

   // slow down to human time frame

   delay(8);
   }

////////////////////////////////////
// move sprite diagonally up
// left

for(;;) {
   if(row < 50) {
      break;
      }

   // move the sprite image

  L1.MoveImage(col--, row--);
  L2.MoveImage(col, row);

   // slow down to human time frame

   delay(8);
   }

setcolor(BLUE);
outtextxy(140, 20, "E");
outtextxy(140, 20, "ENTER Runs Program -- F10 Quits to DOS");

key= K1.Get();

if(key != F10) {
   goto restart_program;
   }

// program has been aborted before it is run
```

Figure 4-1 *Continued.*

```
program_abort:

   // return from graphics mode

   closegraph();

   // free memory allocated during playfield initialization
     U1.Free();
     U2.Free();
     L1.Free();
     L2.Free();
     R1.Free();
     R2.Free();
     P1.Destroy();
}
```

Figure 4-2 *The source code for UGHOST.C.*

```
// ////////////////////////////////////
// ughost.c
//

unsigned char UGHOST1[48*7*8]= {
0, 0, 0, 0, 0, 0, 0,
0, 15, 255, 255, 240, 0, 0,
0, 255, 255, 255, 255, 0, 0,
3, 255, 255, 255, 255, 192, 0,
7, 255, 255, 255, 255, 240, 0,
15, 255, 255, 255, 255, 248, 0,
31, 255, 255, 255, 255, 252, 0,
63, 255, 255, 255, 255, 254, 0,
63, 225, 255, 255, 195, 254, 0,
63, 128, 127, 255, 0, 254, 0,
63, 0, 63, 254, 0, 126, 0,
63, 0, 63, 254, 0, 126, 0,
127, 0, 63, 254, 0, 127, 0,
127, 128, 127, 255, 0, 255, 0,
127, 225, 255, 255, 195, 255, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
```

```
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
127, 255, 255, 255, 255, 255, 0,
63, 255, 255, 255, 255, 254, 0,
1, 255, 15, 248, 127, 192, 0,
0, 254, 7, 240, 63, 128, 0,
0, 254, 7, 240, 63, 128, 0,
0, 254, 7, 240, 63, 128, 0,
0, 254, 7, 240, 63, 128, 0,
0, 254, 7, 240, 63, 128, 0,
0, 254, 7, 240, 63, 128, 0,
0, 254, 7, 240, 63, 128, 0,
0, 254, 7, 240, 63, 128, 0,
0, 254, 7, 240, 63, 128, 0,
1, 255, 15, 248, 127, 192, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0
```

Figure 4-2 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 4-2 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 4-2 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 4-2 *Continued.*

```
0, 30, 0, 0, 60, 0, 0,
0, 63, 0, 0, 126, 0, 0,
0, 30, 0, 0, 60, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 4-2 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 4-2 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 4-2 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0 };

//!
// End of sprite source
//////////////////////////////////////////
```

The source code for LGHOST.C.

Figure 4-3

```
// //////////////////////////////////////
// lghost.c
//

unsigned char LGHOST1[48*7*8]= {
    0, 0, 0, 0, 0, 0, 0,
    0, 15, 255, 255, 240, 0, 0,
    0, 255, 255, 255, 255, 0, 0,
    3, 255, 255, 255, 255, 192, 0,
    7, 255, 255, 255, 255, 240, 0,
    15, 255, 255, 255, 255, 248, 0,
    31, 255, 255, 255, 255, 252, 0,
    63, 255, 255, 255, 255, 254, 0,
    63, 225, 255, 255, 195, 254, 0,
    63, 128, 127, 255, 0, 254, 0,
    63, 0, 63, 254, 0, 126, 0,
    63, 0, 63, 254, 0, 126, 0,
    127, 0, 63, 254, 0, 127, 0,
    127, 128, 127, 255, 0, 255, 0,
    127, 225, 255, 255, 195, 255, 0,
    127, 255, 255, 255, 255, 255, 0,
    127, 255, 255, 255, 255, 255, 0,
    127, 255, 255, 255, 255, 255, 0,
    127, 255, 255, 255, 255, 255, 0,
    127, 255, 255, 255, 255, 255, 0,
    127, 255, 255, 255, 255, 255, 0,
    127, 255, 255, 255, 255, 255, 0,
    127, 255, 255, 255, 255, 255, 0,
    127, 255, 255, 255, 255, 255, 0,
    127, 255, 255, 255, 255, 255, 0,
    127, 255, 255, 255, 255, 255, 0,
    127, 255, 255, 255, 255, 255, 0,
    127, 255, 255, 255, 255, 255, 0,
    127, 255, 255, 255, 255, 255, 0,
    127, 255, 255, 255, 255, 255, 0,
    127, 255, 255, 255, 255, 255, 0,
```

Figure 4-3 *Continued.*

```
63, 255, 255, 255, 255, 254, 0,
1, 255, 15, 248, 127, 192, 0,
0, 254, 7, 240, 63, 128, 0,
0, 254, 7, 240, 63, 128, 0,
0, 255, 7, 240, 63, 128, 0,
0, 127, 3, 248, 31, 128, 0,
0, 63, 129, 252, 15, 192, 0,
0, 31, 192, 254, 7, 224, 0,
0, 15, 224, 127, 3, 240, 0,
0, 7, 240, 63, 129, 248, 0,
0, 3, 252, 31, 192, 254, 0,
0, 7, 254, 63, 225, 255, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 4-3 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 4-3 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 4-3 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 4-3 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 4-3 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 4-3 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 4-4 *The source code for RGHOST.C.*

```
// ////////////////////////////////////
// rghost.c
//

unsigned char RGHOST1[48*7*8]= {
   0, 0, 0, 0, 0, 0, 0,
   0, 15, 255, 255, 240, 0, 0,
   0, 255, 255, 255, 255, 0, 0,
   3, 255, 255, 255, 255, 192, 0,
   7, 255, 255, 255, 255, 240, 0,
   15, 255, 255, 255, 255, 248, 0,
   31, 255, 255, 255, 255, 252, 0,
   63, 255, 255, 255, 255, 254, 0,
   63, 225, 255, 255, 195, 254, 0,
   63, 128, 127, 255, 0, 254, 0,
   63, 0, 63, 254, 0, 126, 0,
   63, 0, 63, 254, 0, 126, 0,
   127, 0, 63, 254, 0, 127, 0,
   127, 128, 127, 255, 0, 255, 0,
   127, 225, 255, 255, 195, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   127, 255, 255, 255, 255, 255, 0,
   63, 255, 255, 255, 255, 254, 0,
   1, 255, 15, 248, 127, 192, 0,
   0, 254, 7, 240, 63, 128, 0,
   0, 254, 7, 240, 63, 128, 0,
   0, 254, 7, 240, 63, 128, 0,
   0, 252, 7, 224, 63, 0, 0,
   1, 248, 15, 192, 126, 0, 0,
   3, 240, 31, 128, 252, 0, 0,
   7, 224, 63, 1, 248, 0, 0,
   15, 192, 126, 3, 240, 0, 0,
   31, 128, 252, 7, 224, 0, 0,
   127, 195, 254, 31, 240, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 4-4 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 4-4 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 4-4 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
```

Figure 4-4 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 4-4 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 4-4 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 4-4 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

5

Simultaneous sprite, missile, & bullet movement

CHAPTER 5

THIS chapter introduces instances of the missile, bullet, and sprite classes in the same program. The missile and bullet data definitions are coded in assembly language. If you have any questions about the assembly source, check your assembler manuals. The basic directives that are used will be supported by most modern assemblers (TASM, MASM, etc.).

Sprites are programmer defined (using the sprite editor presented in the book is the easiest way to design sprites); missiles and bullets are not. Missiles and bullets are predefined shapes of fixed size, and missiles are larger than bullets. Running PROG5-1.EXE, shown in Fig. 5-1, and seeing the sprite, missile, and bullet in motion will do more to describe it than my spewing pixel dimensions. Run it now!

This demonstration program moves a sprite, missile, and bullet smoothly about the screen. The graphic objects move at different rates. Examine the source code presented in Fig. 5-1 carefully. You'll see how the code provides a foundation for the tank game in chapter 8.

Polling

Polling is a technique that can be used in a single-threaded operating system (such as DOS) to control allocation of CPU time to different tasks. Simply, here's why you need to allocate CPU time to smoothly move graphic objects about the screen: In video games, as in real life, one object rarely moves while all other objects remain still. In the tank game presented in chapter 8, the tank moves about the screen while five ghosts take chase.

Imagine that the tank moved for three seconds and then stopped. Once the tank stopped, then ghost 1 would take chase for three seconds and gobble up the tank in one fell swoop. I'm sure you get my point. The tank and all the ghosts need to appear to move on the screen at the same time. That mission is accomplished through CPU polling.

For the tank program in chapter 8, here are the tasks that need to be polled:

- Read the keyboard while not stopping program execution
- Update tank multicolored sprite position
- If missile is launched, update missile position
- Update multicolored ghost 1 position
- Update multicolored ghost 2 position
- Update multicolored ghost 3 position
- Update multicolored ghost 4 position
- Update multicolored ghost 5 position
- If missile and tank (sprite) collide, erase missile and update score

A task such as updating a ghost means that you're moving the position of the ghost a very small amount. When you move all the graphic objects residing in the playfield a small amount on a consistently quick basis, the impact is smooth animation.

Sprite, missile, and bullet movement

Figure 5-1 is the source code for PROG5-1.CPP, which demonstrates a basic polling technique in a single-threaded operating system. This program bounces a multishaped sprite, missile, and bullet around the screen. When one of the moving graphic objects encounters a screen edge, a tiny crash is heard and the graphic object reflects in another direction.

As with the sprite class, creating an instance of the missile class is quite simple. For example:

```
Missile M1( 3 );
```

creates an instance of the missile class named M1. This missile will be drawn on plane 3 and rightfully assume the color mapped to that plane. Once instantiated, you can move the missile class instance by invoking the following missile class method:

```
M1.MoveImage( mcol, mrow );
```

As with the missile class, creating an instance of the bullet class is quite simple. For example:

```
Bullet B1( 2 );
```

creates an instance of the bullet class named B11. This bullet will be drawn on plane 2 and rightfully assume the color mapped to that plane. Once instantiated, you can move the bullet class instance by invoking the following bullet class method:

```
B1.MoveImage( mcol, mrow );
```

That's it. Note that the method for moving the sprite, missile, and bullet class images are identical. This naming convention will allow you to write code without going to the manual every second. All you need to remember is the MoveImage(...) method name and apply that to any of the graphic shape classes.

Figure 5-2 presents the assembly source code for the missile. The assembly .RADIX directive makes declaring graphics object shapes easier in assembly (in my humble opinion) than in C++.

Figure 5-3 presents the assembly source code for the bullet. Note how it's quite similar to that of the missile.

Summary

This chapter presents a demonstration program to move a sprite, missile, and bullet instance smoothly about the screen at the same time. This feat is accomplished by allocating small amounts of CPU time to updating the position of each graphic object. Allocating small amounts of CPU time to different tasks is called *polling*.

Chapter 6 will demonstrate how to move a two-colored tank smoothly on a path by controlling the tank's motion from the keyboard.

The source code for PROG5-1.CPP. Figure 5-1

```
//////////////////////////////////////
//
// prog5-1.cpp
//
// Demonstration of simple movement of
// one sprite, one missile, and one bullet
//

//////////////////////////////////////
//
//  defines
//

#include <graphics.h>
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include "tproto.h"

#include "goclass.h"

//////////////////////////////////////
//
// include sprite created by the
// sprite editor

#include "UGHOST.C"
#include "LGHOST.C"
#include "RGHOST.C"

#define UP_LEFT_MOTION 10
#define UP_RIGHT_MOTION 11
#define DOWN_LEFT_MOTION 12
#define DOWN_RIGHT_MOTION 13

int poll_sprite(int param);
int poll_missile(int param);
int poll_bullet1(int param);
void main(void);
void crash(void);
void crash1(void);
void do_poll_sprite(void);
void do_poll_missile(void);
void do_poll_bullet1(void);

int  row, col, mrow, mcol, mrow2, mcol2, cnt, key;
int max_x, max_y;
```

Figure 5-1 *Continued.*

```
int direction= UP_LEFT_MOTION;
int mdirection= UP_LEFT_MOTION;
int mdirection2= UP_LEFT_MOTION;

// define pointer to class member function

typedef VOID (Bullet::*BMI)(int, int);
typedef VOID (Missile::*MMI)(int, int);
typedef VOID (Sprite::*SMI)(int, int);
typedef VOID (Sound::*ON)(int);
typedef VOID (Sound::*OFF)();

// declare pointers to sprite member functions

BMI BMoveImagePtr;
MMI MMoveImagePtr;
SMI SMoveImagePtr;
ON  SoundOnPtr;
OFF SoundOffPtr;

Sound*  SndPtr;

Bullet*  B1Ptr;
Missile* M1Ptr;
Sprite*  S1Ptr;
Sprite*  S2Ptr;
Sprite*  S3Ptr;

void main()
{
int param;
int gdriver= DETECT, gmode, errorcode;
int midx, midy, mode;
int max_x, max_y, toggle= 0;

  // sound constructor

  Sound Snd(0);

  // set sound on & off member function pointer

  SoundOnPtr=  &Sound::On;
  SoundOffPtr=  &Sound::Off;

  // set SndPtr to instance

  SndPtr= &Snd;

  // keyboard constructor

  Keyboard K1(0);

  initgraph(&gdriver, &gmode, "");
```

```
errorcode= graphresult();

if(errorcode != grOk) {
   printf("Graphics error: %s\n", grapherrormsg(errorcode));
   printf("Press any to return to DOS\n");
   getch();
   exit(1);
   }

// get max X and Y values

max_x= getmaxx();
max_y= getmaxy();

// sprite constructors initialize data

Sprite S1(UGHOST1, 1);
Sprite S2(LGHOST1, 1);
Sprite S3(RGHOST1, 1);

// set sprite member function pointer to MoveImage

SMoveImagePtr= &Sprite::MoveImage;

// set sprite pointers to each sprite instance

S1Ptr= &S1;
S2Ptr= &S2;
S3Ptr= &S3;

 setcolor(4);
 line(0, 0, max_x, 0);
 for(cnt= 0; cnt < 480; cnt++)  {
    line(0, cnt, max_x, cnt);
    }
 line(0, 0, 0, max_y);
 line(0, max_y, max_x,  max_y);
 line(max_x, 0, max_x,  max_y);
 setcolor(0);

PlayField P1(0);

// missile constructor

Missile M1(3);

// set missile member function pointer to MoveImage

MMoveImagePtr= &Missile::MoveImage;

// set sprite pointers to each sprite instance

M1Ptr= &M1;
```

Figure 5-1 *Continued.*

```
   mcol= 20;
   mrow= 20;

  M1.MoveImage(mcol, mrow);

  // missile constructor

  Bullet B1(3);

  // set missile member function pointer to MoveImage

  BMoveImagePtr= &Bullet::MoveImage;

  // set sprite pointers to each sprite instance

  B1Ptr= &B1;

   mcol2= 10;
   mrow2= 10;

  B1.MoveImage(mcol2, mrow2);

   col= 400;
   row= 200;

  S1.MoveImage(col, row);

   for(;;) {
      key= K1.Status();
      switch(key) {
         case F10:
            goto exit_loop;
         }

      delay(2);

      do_poll_bullet1();

      if(!toggle) {
         toggle= 1;
         }
      else {
            toggle= 0;
            do_poll_sprite();
            do_poll_missile();
            do_poll_bullet1();
            }

         do_poll_bullet1();
         do_poll_missile();
         }

  exit_loop:
```

```
    closegraph();

  S1.Free();
  S2.Free();
  S3.Free();
  P1.Destroy();
}

int poll_sprite(int param)
{

   switch(param) {
      case 1:
         if(col > 8) {
                (S2Ptr->*SMoveImagePtr)(col--, row);
            }
         else {
            return 0;
            }
         break;
      case 2:
         if(col < 639 - (7 * 8) ) {
                (S3Ptr->*SMoveImagePtr)(col++, row);
            }
         else {
            return 0;
            }
      break;
   case 3:
      if(row < 479 - 48 - 2) {
             (S1Ptr->*SMoveImagePtr)(col, row++);
         }
      else {
         return 0;
         }
      break;
   case 4:
      if(row > 1) {
             (S1Ptr->*SMoveImagePtr)(col, row--);
         }
      else {
         return 0;
            }
          break;
        }
    return 1;
}

void do_poll_sprite()
{
  if(direction == UP_LEFT_MOTION) {
     if(!poll_sprite(1)) {
        direction= UP_RIGHT_MOTION;
```

Figure 5-1 *Continued.*

```
            poll_sprite(2);
            crash1();
            }
         delay(2);
         if(!poll_sprite(4)) {
            direction= DOWN_LEFT_MOTION;
            poll_sprite(3);
            crash1();
            }
      }

   if(direction == DOWN_LEFT_MOTION) {
      if(!poll_sprite(1)) {
         direction= DOWN_RIGHT_MOTION;
         poll_sprite(2);
         crash1();
         }
      delay(2);
      if(!poll_sprite(3)) {
         direction= UP_LEFT_MOTION;
         poll_sprite(4);
         crash1();
         }
      }

   if(direction == DOWN_RIGHT_MOTION) {
      if(!poll_sprite(2)) {
         direction= DOWN_LEFT_MOTION;
         poll_sprite(1);
         crash1();
         }
      delay(2);
      if(!poll_sprite(3)) {
         direction= UP_RIGHT_MOTION;
         poll_sprite(4);
         crash1();
         }
      }

   if(direction == UP_RIGHT_MOTION) {
      if(!poll_sprite(2)) {
         direction= UP_LEFT_MOTION;
         poll_sprite(1);
         crash1();
         }
      delay(2);
      if(!poll_sprite(4)) {
         direction= DOWN_RIGHT_MOTION;
         poll_sprite(3);
         crash1();
         }
      }
}
```

```
int poll_missile(int param)
{
   switch(param) {
      case 1:
         if(mcol > 8) {
            (M1Ptr->*MMoveImagePtr)(mcol--, mrow);
            }
         else {
            return 0;
            }
      break;
   case 2:
      if(mcol < 639 - (2 * 8) ) {
         (M1Ptr->*MMoveImagePtr)(mcol++, mrow);
         }
      else {
         return 0;
         }
      break;
   case 3:
      if(mrow < 479 - 8) {
         (M1Ptr->*MMoveImagePtr)(mcol, mrow++);
         }
      else {
         return 0;
            }
         break;
      case 4:
         if(mrow > 1) {
            (M1Ptr->*MMoveImagePtr)(mcol, mrow--);
            }
         else {
            return 0;
            }
         break;
      }
   return 1;
}

void do_poll_missile()
{
  if(mdirection == UP_LEFT_MOTION) {
   if(!poll_missile(1)) {
      mdirection= UP_RIGHT_MOTION;
      poll_missile(2);
      crash();
      }
   delay(2);
   if(!poll_missile(4)) {
      mdirection= DOWN_LEFT_MOTION;
      poll_missile(3);
      crash();
      }
   }
```

Figure 5-1 *Continued.*

```
if(mdirection == DOWN_LEFT_MOTION) {
   if(!poll_missile(1)) {
      mdirection= DOWN_RIGHT_MOTION;
      poll_missile(2);
      crash();
      }
   delay(2);
   if(!poll_missile(3)) {
      mdirection= UP_LEFT_MOTION;
      poll_missile(4);
      crash();
      }
   }

if(mdirection == DOWN_RIGHT_MOTION) {
   if(!poll_missile(2)) {
      mdirection= DOWN_LEFT_MOTION;
      poll_missile(1);
      crash();
      }
   delay(2);
   if(!poll_missile(3)) {
      mdirection= UP_RIGHT_MOTION;
      poll_missile(4);
      crash();
      }
   }

if(mdirection == UP_RIGHT_MOTION) {
   if(!poll_missile(2)) {
      mdirection= UP_LEFT_MOTION;
      poll_missile(1);
      crash();
      }
   delay(2);
   if(!poll_missile(4)) {
      mdirection= DOWN_RIGHT_MOTION;
      poll_missile(3);
      crash();
      }
   }
 }

int poll_bullet1(int param)
{
   switch(param) {
      case 1:
         if(mcol2 > 8) {
            (B1Ptr->*BMoveImagePtr)(mcol2--, mrow2);
            }
         else {
            return 0;
            }
```

```
   break;
case 2:
   if(mcol2 < 639 - (2 * 8) ) {
       (B1Ptr->*BMoveImagePtr)(mcol2++, mrow2);
      }
   else {
      return 0;
      }
   break;
case 3:
   if(mrow2 < 479 - 8) {
       (B1Ptr->*BMoveImagePtr)(mcol2, mrow2++);
      }
   else {
      return 0;
      }
   break;
case 4:
   if(mrow2 > 1) {
       (B1Ptr->*BMoveImagePtr)(mcol2, mrow2--);
      }
   else {
      return 0;
      }
   break;
    }
  return 1;
}

void do_poll_bullet1()
{
  if(mdirection2 == UP_LEFT_MOTION) {
     if(!poll_bullet1(1)) {
        mdirection2= UP_RIGHT_MOTION;
        poll_bullet1(2);
        crash();
        }
     if(!poll_bullet1(4)) {
        mdirection2= DOWN_LEFT_MOTION;
        poll_bullet1(3);
        crash();
        }
    }

if(mdirection2 == DOWN_LEFT_MOTION) {
   if(!poll_bullet1(1)) {
      mdirection2= DOWN_RIGHT_MOTION;
      poll_bullet1(2);
      crash();
      }
   if(!poll_bullet1(3)) {
      mdirection2= UP_LEFT_MOTION;
      poll_bullet1(4);
```

Figure 5-1 *Continued.*

```
            crash();
            }
         }

      if(mdirection2 == DOWN_RIGHT_MOTION) {
         if(!poll_bullet1(2)) {
            mdirection2= DOWN_LEFT_MOTION;
            poll_bullet1(1);
            crash();
            }
         if(!poll_bullet1(3)) {
            mdirection2= UP_RIGHT_MOTION;
            poll_bullet1(4);
            crash();
            }
         }

      if(mdirection2 == UP_RIGHT_MOTION) {
         if(!poll_bullet1(2)) {
            mdirection2= UP_LEFT_MOTION;
            poll_bullet1(1);
            crash();
            }
         if(!poll_bullet1(4)) {
            mdirection2= DOWN_RIGHT_MOTION;
            poll_bullet1(3);
            crash();
            }
         }
       }

      int crash_flag=  0;
      int crash1_flag= 0;

      void crash()
      {
        (SndPtr->*SoundOnPtr)(300);
        delay(2);
        (SndPtr->*SoundOffPtr)();
      }

      void crash1()
      {
        (SndPtr->*SoundOnPtr)(1300);
        delay(4);
        (SndPtr->*SoundOffPtr)();
      }
```

The source code for the missile's data. Figure 5-2

```
_missile_data label byte

                .radix 2
                DB 00000000, 00000000 ; 1
                DB 00111000, 00000000
                DB 01111100, 00000000
                DB 01111100, 00000000
                DB 00111000, 00000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 2
                DB 00011100, 00000000
                DB 00111110, 00000000
                DB 00111110, 00000000
                DB 00011100, 00000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 3
                DB 00001110, 00000000
                DB 00011111, 00000000
                DB 00011111, 00000000
                DB 00001110, 00000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 4
                DB 00000111, 00000000
                DB 00001111, 10000000
                DB 00001111, 10000000
                DB 00000111, 00000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 5
                DB 00000011, 10000000
                DB 00000111, 11000000
                DB 00000111, 11000000
                DB 00000011, 10000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 6
                DB 00000001, 11000000
                DB 00000011, 11100000
                DB 00000011, 11100000
                DB 00000001, 11000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 7
                DB 00000000, 11100000
                DB 00000001, 11110000
                DB 00000001, 11110000
                DB 00000000, 11100000
                DB 00000000, 00000000
```

Figure 5-2 *Continued.*

```
DB 00000000, 00000000 ; 8
DB 00000000, 01110000
DB 00000000, 11111000
DB 00000000, 11111000
DB 00000000, 01110000
DB 00000000, 00000000
```

Figure 5-3 *The source code for the bullet's data.*

```
_bullet_data label byte

                .radix 2
                DB 00000000, 00000000 ; 1
                DB 01100000, 00000000
                DB 01100000, 00000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 2
                DB 00110000, 00000000
                DB 00110000, 00000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 3
                DB 00011000, 00000000
                DB 00011000, 00000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 4
                DB 00001100, 00000000
                DB 00001100, 00000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 5
                DB 00000110, 00000000
                DB 00000110, 00000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 6
                DB 00000011, 00000000
                DB 00000011, 00000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 7
                DB 00000001, 10000000
                DB 00000001, 10000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 8
                DB 00000000, 11000000
                DB 00000000, 11000000
                DB 00000000, 00000000
```

6

Controlling a sprite within a track

CHAPTER 6

THE demonstration program in this chapter shows how to move a sprite within a clearly defined track. The player initiates sprite movement by pressing an arrow key in the desired direction. Once the sprite is moving, if the player releases the key the sprite will continue moving in the set direction.

If the player wants to alter the direction of the sprite, he can do so by pressing another arrow key. If the new arrow key directs the sprite in an illegal direction based on its current position, that key press is remembered and the sprite will go in the new direction when the move is legal. Being able to control the sprite from the keyboard lets you "think ahead" with your fingers.

Keeping the sprite in line

Figure 6-1 shows the source code for PROG6-1.CPP. Examine it carefully to see how controlling the sprite via the keyboard works. Figure 6-2 shows the source code for UTANK.C, which was created by the sprite editor. It presents the description of the up-tank sprite.

Figure 6-3 shows the source code listing for DTANK.C, which was created by the sprite editor in less than one second. Here's how: simply take the up-tank image and invert the figure using one of the sprite editor's rotation functions. The sprite creation process is as follows:

❶ Draw the up-tank sprite and save the source to disk.

❷ Invert the image and save it to disk.

❸ Invert the image once again, rotate it clockwise by 90 degrees, and save it to disk.

❹ Click on the rotated by 90 degrees clockwise image twice, and save the rotated by 90 degrees counterclockwise image to disk.

Drawing the original up-tank image will take some time, but once you complete that task, creating the left tank, right tank, and down tank will take just a few moments. Although a tad rough at the edges, the sprite editor is a useful productivity booster.

Figure 6-4 shows the source code for RTANK.C. This source file, created by the sprite editor, presents the description of the right-tank sprite. See Fig. 6-5 for the LTANK.C source code. This was created by the sprite editor, and presents the description of the left-tank sprite.

Summary

The demonstration program PROG6-1.CPP shows you how to run a sprite instance in a maze, while controlling its direction via the keyboard. The sprite editor was used to create a basic tank image comprised of two sprites. Chapter 7 will show you how to shoot a missile from the cannon of the tank via a keyboard press.

The source code for PROG6-1.CPP. Figure 6-1

```
//////////////////////////////////////
//
// prog6-1.cpp
//
// Demonstration of keyboard-controlled
// two-color sprite movement
//

//////////////////////////////////////
//
//  defines
//

#include <graphics.h>
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include "tproto.h"
#include "goclass.h"

#include "utank.c"
#include "dtank.c"
#include "rtank.c"
#include "ltank.c"

#include "go.cpp"
```

Figure 6-1 *Continued.*

```
//////////////////////////////////////////
//
// include sprite created by the
// sprite editor

void init_playfield(void);
int  legal_sprite_move(int param);
void destroy_playfield(void);
void poll_sprite(int param);
void main(void);

#define SPRITE_TRACK          52
#define SPRITE_TRACK_OFFSET   (SPRITE_TRACK / 2)

typedef VOID (Missile::*MMI)(int, int);
typedef VOID (Sprite::*SMI)(int, int);
typedef VOID (Sound::*ON)(int);
typedef VOID (Sound::*OFF)();

SMI SMoveImagePtr;
Sprite*  UT1Ptr;
Sprite*  UT2Ptr;
Sprite*  DT1Ptr;
Sprite*  DT2Ptr;
Sprite*  RT1Ptr;
Sprite*  RT2Ptr;
Sprite*  LT1Ptr;
Sprite*  LT2Ptr;

int  row, col, cnt, key;
int max_x, max_y;
int midx, midy, mode;

void main()
{
int param, cnt;
int gdriver= DETECT, gmode, errorcode;
int old_row, old_col;
char location_buf[25];
int param_case, show_row_col= 0;

initgraph(&gdriver, &gmode, "");

errorcode= graphresult();

if(errorcode != grOk) {
   printf("Graphics error: %s\n", grapherrormsg(errorcode));
   printf("Press any to return to DOS\n");
   getch();
   exit(1);
   }

// get max X and Y values
```

```
max_x= getmaxx();
max_y= getmaxy();

// calculate midpoints

midx= max_x / 2;
midy= max_y /2;

// outer box

rectangle(0, 0, max_x, max_y);

// upper left inner box

rectangle(52,
          52,
          midx - SPRITE_TRACK_OFFSET,
          midy - SPRITE_TRACK_OFFSET);

// upper right inner box

rectangle(midx + SPRITE_TRACK_OFFSET,
          SPRITE_TRACK,
          max_x - SPRITE_TRACK,
          midy - SPRITE_TRACK_OFFSET);

 // lower left inner box

 rectangle(SPRITE_TRACK,
           midy + SPRITE_TRACK_OFFSET,
           midx - SPRITE_TRACK_OFFSET,
           max_y - SPRITE_TRACK);

 // lower right inner box

 rectangle(midx + SPRITE_TRACK_OFFSET,
           midy + SPRITE_TRACK_OFFSET,
           max_x - SPRITE_TRACK,
           max_y - SPRITE_TRACK);

PlayField P1(0);
Keyboard K1(0);
 col= 2;
 row= 100;

Sprite UT1(UTANK1, 1);
Sprite UT2(UTANK2, 4);

Sprite DT1(DTANK1, 1);
Sprite DT2(DTANK2, 4);

Sprite RT1(RTANK1, 1);
```

Figure 6-1 *Continued.*

```
Sprite RT2(RTANK2, 4);

Sprite LT1(LTANK1, 1);
Sprite LT2(LTANK2, 4);

DT2.MoveImage(col, row);
DT1.MoveImage(col, row);

// set sprite member function pointer to MoveImage

SMoveImagePtr= &Sprite::MoveImage;

// set sprite pointers to each sprite instance

UT1Ptr= &UT1;
UT2Ptr= &UT2;
DT1Ptr= &DT1;
DT2Ptr= &DT2;
RT1Ptr= &RT1;
RT2Ptr= &RT2;
LT1Ptr= &LT1;
LT2Ptr= &LT2;

 setbkcolor(DARKGRAY);

 for(;;) {

if(((row != old_row) || (col != old_col)) && (show_row_col)) {
   memset(location_buf, 0, 25);
   setwritemode(COPY_PUT);
   setcolor(DARKGRAY);
   line(90, 90, 90 + (40 * 8), 10);
   for(cnt= 10; cnt < 30; cnt++) {
      line(90, 90 + cnt, 90 + (40 * 8), 90 + cnt);
      }
   sprintf(location_buf,"Row= %3d  Col= %3d", row, col);
   moveto(100, 100);
   setcolor(WHITE);
   outtext(location_buf);

   old_row= row;
   old_col= col;
   }

key= K1.Status();

switch(key) {
   case LEFT_ARROW:
      param= 1;
      break;
   case RIGHT_ARROW:
      param= 2;
      break;
   case UP_ARROW:
      param= 4;
```

```
      break;
   case DOWN_ARROW:
      param= 3;
      break;
   case F1:
      show_row_col= 1;
      break;
   case F2:
      show_row_col= 0;
      break;
   case ENTER:
      param= 0;
      break;
   case F10:
      goto exit_loop;
   }

delay(6);

if(legal_sprite_move(param)) {
   poll_sprite(param);
   param_case= param;
   }
else {
   if(legal_sprite_move(param_case)) {
      poll_sprite(param_case);
         }
      }

   }

exit_loop:

  P1.Destroy();

   closegraph();

}

void poll_sprite(int param)
{
switch(param) {
   // left tank motion
   case 1:
     (LT1Ptr->*SMoveImagePtr)(col--, row);
     (LT2Ptr->*SMoveImagePtr)(col, row);
      break;

   // right tank motion
   case 2:
     (RT1Ptr->*SMoveImagePtr)(col++, row);
     (RT2Ptr->*SMoveImagePtr)(col, row);
      break;
```

Figure 6-1 *Continued.*

```
        // down tank motion
        case 3:
          (DT1Ptr->*SMoveImagePtr)(col, row++);
          (DT2Ptr->*SMoveImagePtr)(col, row);
           break;
        // up tank motion
        case 4:
          (UT1Ptr->*SMoveImagePtr)(col, row--);
          (UT2Ptr->*SMoveImagePtr)(col, row);
        }
}

int legal_sprite_move(int param)
{

   switch(param) {
      // left tank motion
      case 1:
         if((col > 2) && ((row == 2) ¦¦
                          (row == 428) ¦¦
                          (row == midy - SPRITE_TRACK_OFFSET))) {
            return 1;
            }
         break;

        // right tank motion
        case 2:
           if((col < 590) && ((row == 2)   ¦¦
                              (row == 428) ¦¦
                              (row == midy - SPRITE_TRACK_OFFSET))) {
              return 1;
              }
           break;
        // down tank motion
        case 3:
           if((row < 428) && ((col == 2)  ¦¦
                              (col == 590)¦¦
                              (col == midx - SPRITE_TRACK_OFFSET))) {
              return 1;
              }
           break;
           // up tank motion
           case 4:
              if((row > 2)  && ((col == 2)   ¦¦
                                (col == 590) ¦¦
                                (col == midx - SPRITE_TRACK_OFFSET))) {
                 return 1;
                 }
              break;
           }
      return 0;
}
```

The source code for UTANK.C.

Figure 6-2

```
////////////////////////////////////////
// UTANK.c
//

unsigned char UTANK1[48*7*8]= {
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
```

Figure 6-2 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-2 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-2 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-2 *Continued.*

```
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
```

```
0, 63, 255, 255, 252, 0, 0,
0, 127, 255, 255, 254, 0, 0,
0, 63, 255, 255, 252, 0, 0,
0, 127, 255, 255, 254, 0, 0,
0, 63, 255, 255, 252, 0, 0,
0, 127, 255, 255, 254, 0, 0,
0, 31, 255, 255, 248, 0, 0,
0, 15, 255, 255, 240, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0
```

Figure 6-2 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-2 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-2 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
```

Figure 6-3 *Continued.*

```
127, 128, 0, 0, 1, 255, 0,
127, 128, 0, 0, 1, 255, 0,
127, 128, 0, 0, 1, 255, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-3 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-3 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-3 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0 };
unsigned char DTANK2[48*7*8]= {
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 15, 255, 255, 240, 0, 0,
   0, 31, 255, 255, 248, 0, 0,
   0, 127, 255, 255, 254, 0, 0,
   0, 63, 255, 255, 252, 0, 0,
   0, 127, 255, 255, 254, 0, 0,
   0, 63, 255, 255, 252, 0, 0,
   0, 127, 255, 255, 254, 0, 0,
   0, 63, 255, 255, 252, 0, 0,
   0, 127, 255, 255, 254, 0, 0,
   0, 63, 255, 255, 252, 0, 0,
   0, 127, 255, 255, 254, 0, 0,
   0, 63, 255, 255, 252, 0, 0,
   0, 127, 255, 255, 254, 0, 0,
   0, 31, 255, 255, 248, 0, 0,
   0, 15, 255, 255, 240, 0, 0,
   0, 0, 3, 192, 0, 0, 0,
   0, 0, 3, 192, 0, 0, 0,
   0, 0, 3, 192, 0, 0, 0,
   0, 0, 3, 192, 0, 0, 0,
   0, 0, 3, 192, 0, 0, 0,
   0, 0, 3, 192, 0, 0, 0,
   0, 0, 3, 192, 0, 0, 0,
   0, 0, 3, 192, 0, 0, 0,
   0, 0, 3, 192, 0, 0, 0,
   0, 0, 3, 192, 0, 0, 0,
   0, 0, 3, 192, 0, 0, 0,
   0, 0, 7, 224, 0, 0, 0,
   0, 0, 5, 160, 0, 0, 0,
```

Figure 6-3 *Continued.*

```
0, 0, 5, 160, 0, 0, 0,
0, 0, 7, 224, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-3 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-3 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-3 *Continued.*

```
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
3, 255, 255, 254, 0, 0, 0,
7, 255, 255, 255, 0, 0, 0,
15, 255, 255, 255, 128, 0, 0,
15, 255, 255, 255, 128, 0, 0,
15, 255, 255, 255, 128, 0, 0,
15, 255, 255, 255, 128, 0, 0,
7, 255, 255, 255, 0, 0, 0,
3, 255, 255, 254, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0
```

Figure 6-4 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-4 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-4 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-4 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 21, 85, 64, 0, 0, 0,
0, 31, 255, 192, 0, 0, 0,
0, 63, 255, 224, 0, 0, 0,
0, 127, 255, 240, 0, 0, 0,
0, 127, 255, 240, 0, 0, 0,
0, 127, 255, 240, 0, 0, 0,
0, 127, 255, 240, 0, 0, 0,
0, 127, 255, 240, 0, 0, 0,
0, 127, 255, 240, 0, 0, 0,
0, 127, 255, 240, 0, 0, 0,
0, 127, 255, 240, 0, 0, 0,
0, 127, 255, 240, 0, 30, 0,
0, 127, 255, 255, 255, 242, 0,
0, 127, 255, 255, 255, 254, 0,
0, 127, 255, 255, 255, 254, 0,
0, 127, 255, 255, 255, 242, 0,
0, 127, 255, 240, 0, 30, 0,
0, 127, 255, 240, 0, 0, 0,
0, 127, 255, 240, 0, 0, 0,
0, 127, 255, 240, 0, 0, 0,
0, 127, 255, 240, 0, 0, 0,
0, 127, 255, 240, 0, 0, 0,
0, 127, 255, 240, 0, 0, 0,
0, 127, 255, 240, 0, 0, 0,
0, 63, 255, 224, 0, 0, 0,
0, 31, 255, 192, 0, 0, 0,
0, 21, 85, 64, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-4 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-4 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-4 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0 };

//!
// End of sprite source
/////////////////////////////////////////
```

The source code for LTANK.C.

Figure 6-5

```
/////////////////////////////////////////
// LTANK.c
//

unsigned char LTANK1[48*7*8]= {
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 127, 255, 255, 192, 0,
   0, 0, 255, 255, 255, 224, 0,
   0, 1, 255, 255, 255, 240, 0,
   0, 1, 255, 255, 255, 240, 0,
   0, 1, 255, 255, 255, 240, 0,
   0, 0, 255, 255, 255, 224, 0,
   0, 0, 127, 255, 255, 192, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0
```

Figure 6-5 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 127, 255, 255, 192, 0,
0, 0, 255, 255, 255, 224, 0,
0, 1, 255, 255, 255, 240, 0,
0, 1, 255, 255, 255, 240, 0,
0, 1, 255, 255, 255, 240, 0,
0, 1, 255, 255, 255, 240, 0,
0, 1, 255, 255, 255, 240, 0,
0, 0, 255, 255, 255, 224, 0,
0, 0, 127, 255, 255, 192, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-5 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-5 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-5 *Continued.*

```
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 15, 255, 254, 0, 0,
0, 0, 15, 255, 254, 0, 0,
0, 0, 15, 255, 254, 0, 0,
0, 0, 15, 255, 254, 0, 0,
120, 0, 15, 255, 254, 0, 0,
79, 255, 255, 255, 254, 0, 0,
127, 255, 255, 255, 254, 0, 0,
127, 255, 255, 255, 254, 0, 0,
79, 255, 255, 255, 254, 0, 0,
120, 0, 15, 255, 254, 0, 0,
0, 0, 15, 255, 254, 0, 0,
0, 0, 15, 255, 254, 0, 0,
0, 0, 15, 255, 254, 0, 0,
0, 0, 15, 255, 254, 0, 0,
0, 0, 15, 255, 254, 0, 0,
0, 0, 15, 255, 254, 0, 0,
0, 0, 15, 255, 254, 0, 0,
0, 0, 15, 255, 254, 0, 0,
0, 0, 15, 255, 254, 0, 0,
0, 0, 7, 255, 252, 0, 0,
0, 0, 3, 255, 248, 0, 0,
0, 0, 2, 170, 168, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0
```

Figure 6-5 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-5 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

Figure 6-5 *Continued.*

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
```

7

Launching a missile from a moving sprite

CHAPTER 7

LAUNCHING a missile is different from just moving a missile. When you launch a missile, the start location of the launch is relative to the launcher's (sprite's) position. Second, the speed of the missile that has been launched is influenced by the speed and direction of the sprite.

Launching a missile

The PROG7-1.EXE demonstration program uses a space bar to launch a missile from the tank's cannon. The direction of the missile is determined by the tank's direction. The speed of the missile is far greater than the tank.

In PROG7-1.EXE, once the missile is launched it travels unfettered until a boundary is reached. Once the boundary is reached, a crack is heard and then the missile is removed. Examine the source code, shown in Fig. 7-1, to see where that happens. This program demonstrates how to launch a missile from a moving sprite-based tank by pressing the keyboard's space bar.

Also examine the source code to see where the missile launch occurs and understand why you can't launch another missile until the first missile is removed from the playfield. If you don't like the fact that only one missile at a time can travel on the playfield, then change the code in PROG7-1.CPP to accommodate more than one missile being able to move on the playfield at one time.

Summary

This chapter modifies the tank program in chapter 6 so a missile can be launched from the tank via a space-bar press. Note that the missile travels faster than the tank and travels the same direction in which the tank is travelling.

Chapter 8 will present the workings to a simple shoot-the-chasing-ghosts-in-a-maze game. Polling is expanded to account for the motion of five chasing ghosts, along with a tank armed with a missile launcher.

Figure 7-1

The source code for PROG7-1.CPP.

```
//////////////////////////////////////
//
// prog7-1.cpp
//
// Demonstration of keyboard-controlled
// two-color sprite movement with missile
// launch from keyboard
//
//////////////////////////////////////
//
//  defines
//

#include <graphics.h>
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include "tproto.h"

#include "goclass.h"

#include "utank.c"
#include "dtank.c"
#include "rtank.c"
#include "ltank.c"

#include "go.cpp"

typedef VOID (Missile::*ME)();
typedef VOID (Missile::*MMI)(int, int);
typedef VOID (Sprite::*SMI)(int, int);
typedef VOID (Sound::*ON)(int);
typedef VOID (Sound::*OFF)();

ME  MEraseImagePtr;
MMI MMoveImagePtr;

SMI SMoveImagePtr;
Sprite*  UT1Ptr;
Sprite*  UT2Ptr;
Sprite*  DT1Ptr;
Sprite*  DT2Ptr;
Sprite*  RT1Ptr;
Sprite*  RT2Ptr;
Sprite*  LT1Ptr;
Sprite*  LT2Ptr;

ON  SoundOnPtr;
OFF SoundOffPtr;
```

Figure 7-1 *Continued.*

```
Missile* M1Ptr;
Sound*  SndPtr;

////////////////////////////////////////
//
// include sprite created by the
// sprite editor

void crash(void);
void crash1(void);
void set_missile_position(int param);
void launch_missile(int missile_direction);
int  legal_sprite_move(int param);
void poll_sprite(int param);
void main(void);

#define SPRITE_TRACK         52
#define SPRITE_TRACK_OFFSET  (SPRITE_TRACK / 2)

int missile_direction;
int missile_launched= 0;
int missile_row, missile_col;
int  row, col, cnt, key;
int max_x, max_y;
int midx, midy, mode;

void main()
{
int param, cnt;
int gdriver= DETECT, gmode, errorcode;
int old_row, old_col;
char location_buf[25];
int param_case, show_row_col= 0;

   initgraph(&gdriver, &gmode, "");

   errorcode= graphresult();

   if(errorcode != grOk) {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any to return to DOS\n");
      getch();
      exit(1);
      }

// sound constructor

Sound Snd(0);

// set sound on & off member function pointer

SoundOnPtr=  &Sound::On;
SoundOffPtr=  &Sound::Off;
```

```
// keyboard constructor
Keyboard K1(0);
 setbkcolor(DARKGRAY);
 // get max X and Y values
max_x= getmaxx();
max_y= getmaxy();

// calculate midpoints

midx= max_x / 2;
midy= max_y /2;

// outer box

rectangle(0, 0, max_x, max_y);

// upper left inner box

rectangle(52,
          52,
          midx - SPRITE_TRACK_OFFSET,
          midy - SPRITE_TRACK_OFFSET);

// upper right inner box

rectangle(midx + SPRITE_TRACK_OFFSET,
          SPRITE_TRACK,
          max_x - SPRITE_TRACK,
          midy - SPRITE_TRACK_OFFSET);

// lower left inner box

rectangle(SPRITE_TRACK,
          midy + SPRITE_TRACK_OFFSET,
          midx - SPRITE_TRACK_OFFSET,
          max_y - SPRITE_TRACK);

 // lower right inner box

 rectangle(midx + SPRITE_TRACK_OFFSET,
           midy + SPRITE_TRACK_OFFSET,
           max_x - SPRITE_TRACK,
           max_y - SPRITE_TRACK);

PlayField P1(0);

 col= 2;
 row= 100;

// missile constructor
```

Figure 7-1 *Continued.*

```
Missile M1(3);

// set missile member function pointer to MoveImage

MMoveImagePtr= &Missile::MoveImage;
MEraseImagePtr= &Missile::EraseImage;

// set sprite pointers to each sprite instance

M1Ptr= &M1;

Sprite UT1(UTANK1, 1);
Sprite UT2(UTANK2, 4);

Sprite DT1(DTANK1, 1);
Sprite DT2(DTANK2, 4);

Sprite RT1(RTANK1, 1);
Sprite RT2(RTANK2, 4);

Sprite LT1(LTANK1, 1);
Sprite LT2(LTANK2, 4);

DT2.MoveImage(col, row);
DT1.MoveImage(col, row);

// set sprite member function pointer to MoveImage

SMoveImagePtr= &Sprite::MoveImage;

// set sprite pointers to each sprite instance

UT1Ptr= &UT1;
UT2Ptr= &UT2;
DT1Ptr= &DT1;
DT2Ptr= &DT2;
RT1Ptr= &RT1;
RT2Ptr= &RT2;
LT1Ptr= &LT1;
LT2Ptr= &LT2;

for(;;) {

   if(((row != old_row) || (col != old_col)) && (show_row_col)) {
      memset(location_buf, 0, 25);
      setwritemode(COPY_PUT);
      setcolor(DARKGRAY);
      line(90, 90, 90 + (40 * 8), 10);
      for(cnt= 10; cnt < 30; cnt++) {
         line(90, 90 + cnt, 90 + (40 * 8), 90 + cnt);
         }
      sprintf(location_buf,"Row= %3d  Col= %3d", row, col);
```

```
moveto(100, 100);
setcolor(WHITE);
outtext(location_buf);

old_row= row;
old_col= col;
}

key= K1.Status();
switch(key) {
   case SPACE:
      if(!missile_launched) {
         set_missile_position(param_case);
         missile_launched= 1;
         }
      break;
   case LEFT_ARROW:
      param= 1;
      break;
   case RIGHT_ARROW:
      param= 2;
      break;
   case UP_ARROW:
      param= 4;
      break;
   case DOWN_ARROW:
      param= 3;
      break;
   case F1:
      show_row_col= 1;
      break;
   case F2:
      show_row_col= 0;
      break;
   case ENTER:
      param= 0;
      break;
   case F10:
      goto exit_loop;
   }

delay(6);

if(missile_launched) {
  launch_missile(missile_direction);
  launch_missile(missile_direction);
   }

if(legal_sprite_move(param)) {
   poll_sprite(param);
   param_case= param;
   }
else {
```

Figure 7-1 *Continued.*

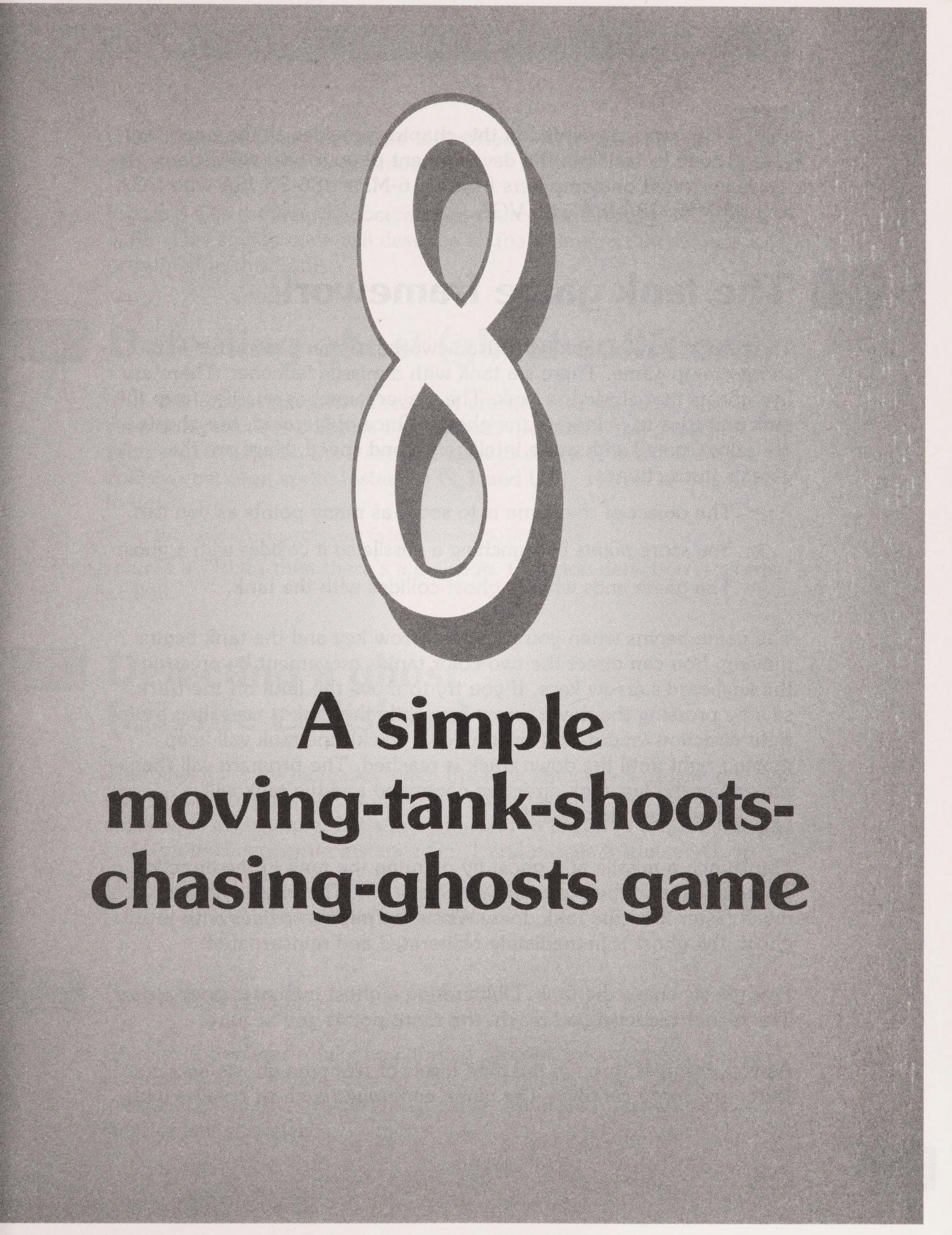
```
            if(legal_sprite_move(param_case)) {
               poll_sprite(param_case);
               }
            }

         if(missile_launched) {
            launch_missile(missile_direction);
            }

         }

exit_loop:
  P1.Destroy();

   closegraph();

}

VOID set_missile_position(int param)
{
   crash1();
   switch(param) {
   // left
   case  1:
      missile_col= col - 7;
      missile_row= row + SPRITE_TRACK_OFFSET - 4;
      missile_direction= param;
      missile_launched= 1;
      break;
   // right
   case  2:
      missile_col= col + SPRITE_TRACK  - 4;
      missile_row= row + SPRITE_TRACK_OFFSET - 4;
      missile_direction= param;
      missile_launched= 1;
      break;
   // down
   case  3:
      missile_col= col + SPRITE_TRACK_OFFSET - 6;
      missile_row= row + SPRITE_TRACK - 4;
      missile_direction= param;
      missile_launched= 1;
      break;
   // up
   case  4:
      missile_col= col + SPRITE_TRACK_OFFSET - 6;
      missile_row= row - 6;
      missile_direction= param;
      missile_launched= 1;
      break;
   }
```

```
        (M1Ptr->*MMoveImagePtr)(missile_col, missile_row);
}

void launch_missile(int param)
{
   switch(param) {
      // left missile motion
      case 1:
      if(missile_col < 16) {
         missile_launched= 0;
          (M1Ptr->*MEraseImagePtr)();
         crash();
         return;
         }
        (M1Ptr->*MMoveImagePtr)(missile_col--, missile_row);
      break;
// right missile motion
case 2:
      if(missile_col > max_x - 16) {
         missile_launched= 0;
          (M1Ptr->*MEraseImagePtr)();
         crash();
         return;
         }
        (M1Ptr->*MMoveImagePtr)(missile_col++, missile_row);
      break;
// down missile motion
case 3:
      if(missile_row > max_y - 16) {
         missile_launched= 0;
          (M1Ptr->*MEraseImagePtr)();
         crash();
         return;
         }
        (M1Ptr->*MMoveImagePtr)(missile_col, missile_row++);
      break;
// up missile motion
case 4:
      if(missile_row < 16) {
         missile_launched= 0;
          (M1Ptr->*MEraseImagePtr)();
         crash();
         return;
         }
           (M1Ptr->*MMoveImagePtr)(missile_col, missile_row--);
         break;
      }
}

void poll_sprite(int param)
{

   switch(param) {
```

Figure 7-1 *Continued.*

```
// left tank motion
case 1:
  (LT1Ptr->*SMoveImagePtr)(col--, row);
  (LT2Ptr->*SMoveImagePtr)(col, row);
   break;

// right tank motion
case 2:
  (RT1Ptr->*SMoveImagePtr)(col++, row);
  (RT2Ptr->*SMoveImagePtr)(col, row);
   break;
// down tank motion
case 3:
  (DT1Ptr->*SMoveImagePtr)(col, row++);
  (DT2Ptr->*SMoveImagePtr)(col, row);
   break;
// up tank motion
case 4:
  (UT1Ptr->*SMoveImagePtr)(col, row--);
  (UT2Ptr->*SMoveImagePtr)(col, row);
   break;
}

int legal_sprite_move(int param)
{

   switch(param) {
      // left tank motion
      case 1:
         if((col > 2) && ((row == 2) ¦¦
                          (row == 428) ¦¦
                          (row == midy - SPRITE_TRACK_OFFSET))) {
            return 1;
            }
         break;

      // right tank motion
      case 2:
      if((col < 590) && ((row == 2)   ¦¦
                         (row == 428) ¦¦
                         (row == midy - SPRITE_TRACK_OFFSET))) {
         return 1;
         }
      break;
// down tank motion
case 3:
   if((row < 428) && ((col == 2)  ¦¦
                      (col == 590)¦¦
                      (col == midx - SPRITE_TRACK_OFFSET + 2))) {
      return 1;
```

```
        }
    break;
// up tank motion
case 4:
    if((row > 2)  && ((col == 2)    ||
                      (col == 590)  ||
                      (col == midx - SPRITE_TRACK_OFFSET + 2))) {
        return 1;
        }
        break;
      }
    return 0;
}

void crash()
{
  (SndPtr->*SoundOnPtr)(300);
  delay(2);
  (SndPtr->*SoundOffPtr)();
}

void crash1()
{
  (SndPtr->*SoundOnPtr)(1300);
  delay(4);
  (SndPtr->*SoundOffPtr)();
}
```



A simple moving-tank-shoots-chasing-ghosts game

CHAPTER 8

THE game presented in this chapter provides all the necessary code to facilitate the development of your own video game. It has been tested on computers from a 16-MHz 386-SX ISA with VGA to a 33-MHz 486 ISA with VGA.

The tank game framework

The demonstration tank game framework has many elements of a shoot-em-up game. There's a tank with a missile launcher. There are five ghosts that chase the tank. The player launches missiles from the tank and tries to obliterate the ghosts. Once obliterated, the ghosts are reincarnated with more intelligence and speed. Here are the game's instructions:

- The object of the game is to score as many points as you can.
- You score points by launching a missile so it collides with a ghost.
- The game ends when a ghost collides with the tank.

The game begins when you press an arrow key and the tank begins moving. You can direct the two-color tank's movement by pressing the keyboard's arrow keys. If you try to move the tank off the track, say, by pressing the down arrow key while the tank is travelling in the right direction (and it isn't over a down track), the tank will keep moving right until the down track is reached. The program will then remember the last tank direction command and the tank will immediately turn downward.

You launch a missile at a ghost by pointing the tank in the direction of the ghost and then pressing the space bar. The missile travels much faster than the tank does. When the missile collides with the ghost, the ghost is immediately obliterated and reincarnated.

Five ghosts chase the tank. Obliterating a ghost increases your score. The higher the level you reach, the more points you acquire.

As you progress through the nine levels of play, the ghosts become faster and more cunning. The game ends when a ghost collides with

your tank, and your final score is displayed. Remember, begin the game by starting the tank's motion.

Figure 8-1 is the heavily documented source code listing, PROG8-1 .CPP. This source code will describe all the animation techniques presented in this book.

Detecting ghost (ghost collisions)

The tank game ends when the ghost crashes into the tank. The current locations are always held in both the tank's and ghosts' private data objects. Let's say you want to check to see if there's a collision between sprite instances G_1 and UT1. Here's all you need to do. If:

```
(G_1.SpriteCollision(UT1.GetInfo(SInfoPtr)) )
```

returns a TRUE, then there's a collision. Collision detection is as easy as that.

Detecting a ghost (missile collisions)

The player's score increases when the tank launches a missile and the missile then collides with a ghost. Detecting a sprite and missile collision is also simple. Let's say you want to check to see if there's a collision between sprite instance G5_1 and missile instance M1. If:

```
(G5_1.MissileCollision(M1.GetInfo(MInfoPtr )))
```

returns a TRUE, then there's a collision.

Summary

Chapter 8 presents a fully functioning prototype for a shoot-'em-up computer game. A tank tries to obliterate ghosts by launching a missile at them. Every time the ghost is obliterated, it's reincarnated smarter and stronger.

Chapter 8

Chapter 8 ends part one of this book. Part two contains the source code for the sprite management library, and part three presents the C and assembly source code for the rudimentary sprite editor.

Figure 8-1 *The source code for PROG8-1.CPP.*

```
//////////////////////////////////////
//
// prog8-1.cpp
//
// Demonstration of keyboard-controlled
// two-color sprite movement with missile
// launch from keyboard and chasing ghosts
//

//////////////////////////////////////
//
//  defines
//

#include <graphics.h>
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include "tproto.h"
#include "goclass.h"

#include "utank.c"
#include "dtank.c"
#include "rtank.c"
#include "ltank.c"
#include "ughost.c"

#include "go.cpp"

//////////////////////////////////////
//
// include sprite created by the
// sprite editor

void remove_sprites(void);
void explosion(void);
void peep(void);
void print_missiles_used(void);
void erase_message_window(void);
void set_level(void);
int  game_over(void);
void init_game(void);
```

```
void print_level(void);
void print_score(int hit);
void crash(void);
void crash1(void);
void set_active_tank(int param);
void set_missile_position(int param);
void launch_missile(int param);
void init_playfield(void);
void launch_ghost1(int row, int col);
void launch_ghost2(int row, int col);
void launch_ghost3(int row, int col);
void launch_ghost4(int row, int col);
void launch_ghost5(int row, int col);
void ghost1_move(void);
void ghost2_move(void);
void ghost3_move(void);
void ghost4_move(void);
void ghost5_move(void);
int  legal_sprite_move(int param);
void poll_sprite(int param);
void main(void);

#define SPRITE_TRACK          64    // track width
#define SPRITE_TRACK_OFFSET   (SPRITE_TRACK / 2)
#define D_T                   1     // down tank
#define U_T                   2     // up tank
#define R_T                   3     // right tank
#define L_T                   4     // left tank
#define LEVEL_1               1
#define LEVEL_2               2
#define LEVEL_3               3
#define LEVEL_4               4
#define LEVEL_5               5
#define LEVEL_6               6
#define LEVEL_7               7
#define LEVEL_8               8
#define LEVEL_9               9

#define DELAY8_MS             8
#define DELAY7_MS             7
#define DELAY6_MS             6
#define DELAY5_MS             5
#define DELAY4_MS             4
#define DELAY3_MS             3
#define DELAY2_MS             2

typedef INT  (Keyboard::*KEY)();
typedef VOID (Sprite::*SE)();
typedef VOID (Missile::*ME)();
typedef VOID (Missile::*MMI)(int, int);
typedef VOID (Sprite::*SMI)(int, int);
typedef VOID (Sound::*ON)(int);
typedef VOID (Sound::*OFF)();
```

Figure 8-1 *Continued.*

```
SE  SEraseImagePtr;
ME  MEraseImagePtr;
MMI MMoveImagePtr;
KEY KGetPtr;

SMI SMoveImagePtr;
Sprite*  UT1Ptr;
Sprite*  UT2Ptr;
Sprite*  DT1Ptr;
Sprite*  DT2Ptr;
Sprite*  RT1Ptr;
Sprite*  RT2Ptr;
Sprite*  LT1Ptr;
Sprite*  LT2Ptr;
Sprite*  G_1Ptr;
Sprite*  G_2Ptr;
Sprite*  G2_1Ptr;
Sprite*  G2_2Ptr;
Sprite*  G3_1Ptr;
Sprite*  G3_2Ptr;
Sprite*  G4_1Ptr;
Sprite*  G4_2Ptr;
Sprite*  G5_1Ptr;
Sprite*  G5_2Ptr;
Sprite*  G6_1Ptr;
Sprite*  G6_2Ptr;

ON  SoundOnPtr;
OFF SoundOffPtr;

Missile* M1Ptr;
Sound*   SndPtr;
Keyboard*  KeyPtr;

int peep_flag= 0;
int missiles_used= 0;
int delay_value= DELAY8_MS;
int active_tank;
int difficulty_level= LEVEL_1;
long total_score= 0;
long adjusted_score= 0;
int score= 0;
int center_sprite_x, center_sprite_y;
int ghost1_direction= 1;
int ghost2_direction= 1;
int ghost3_direction= 1;
int ghost4_direction= 1;
int ghost5_direction= 1;
int ghost1_launched= 0;
int ghost2_launched= 0;
int ghost3_launched= 0;
int ghost4_launched= 0;
```

```
int ghost5_launched= 0;
int missile_direction;
int missile_launched= 0;
int missile_row, missile_col;
int  row, col, g1_row, g1_col, cnt, key;
int row2, col2, g2_row, g2_col;
int row3, col3, g3_row, g3_col;
int row4, col4, g4_row, g4_col;
int row5, col5, g5_row, g5_col;
int max_x, max_y;
int midx, midy, mode;

void main()
{
int param, cnt, val;
int gdriver= DETECT, gmode, errorcode;
int old_row, old_col;
char location_buf[25];
int param_case;
MISP MInfoPtr;
SISP SInfoPtr;
   initgraph(&gdriver, &gmode, "");

   errorcode= graphresult();

 if(errorcode != grOk) {
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any to return to DOS\n");
    getch();
    exit(1);
    }

// sound constructor

Sound Snd(0);

// set sound on & off member function pointer

SoundOnPtr=  &Sound::On;
SoundOffPtr=  &Sound::Off;

// keyboard constructor

Keyboard K1(0);

 setbkcolor(DARKGRAY);

 // get max X and Y values

 max_x= getmaxx();
 max_y= getmaxy();

 // calculate midpoints
```

Figure 8-1 *Continued.*

```
midx= max_x / 2;
midy= max_y /2;

// outer box

rectangle(0, 0, max_x, max_y);

// upper left inner box

rectangle(SPRITE_TRACK,
          SPRITE_TRACK,
          midx - SPRITE_TRACK_OFFSET,
          midy - SPRITE_TRACK_OFFSET - 1);

// upper right inner box

rectangle(midx + SPRITE_TRACK_OFFSET,
          SPRITE_TRACK,
          max_x - SPRITE_TRACK,
          midy - SPRITE_TRACK_OFFSET - 1);

// lower left inner box

rectangle(SPRITE_TRACK,
          midy + SPRITE_TRACK_OFFSET,
          midx - SPRITE_TRACK_OFFSET,
          max_y - SPRITE_TRACK);

 // lower right inner box

 rectangle(midx + SPRITE_TRACK_OFFSET,
          midy + SPRITE_TRACK_OFFSET,
          max_x - SPRITE_TRACK,
          max_y - SPRITE_TRACK);

PlayField P1(0);

// Missile constructor

Missile M1(3);

// set missile, keyboard member function pointer to MoveImage

KGetPtr= &Keyboard::Get;
MMoveImagePtr= &Missile::MoveImage;
MEraseImagePtr= &Missile::EraseImage;

// set pointers to each keyboard and missile instance

KeyPtr= &K1;
M1Ptr= &M1;
```

```
 center_sprite_x= midx - SPRITE_TRACK_OFFSET + 8;
 center_sprite_y= midy - SPRITE_TRACK_OFFSET + 4;

Sprite UT1(UTANK1, 1);
Sprite UT2(UTANK2, 4);

Sprite DT1(DTANK1, 1);
Sprite DT2(DTANK2, 4);

Sprite RT1(RTANK1, 1);
Sprite RT2(RTANK2, 4);

Sprite LT1(LTANK1, 1);
Sprite LT2(LTANK2, 4);

Sprite G_1(UGHOST1, 2);
Sprite G_2(UGHOST2, 4);
Sprite G2_1(UGHOST1, 2);
Sprite G2_2(UGHOST2, 4);
Sprite G3_1(UGHOST1, 2);
Sprite G3_2(UGHOST2, 4);
Sprite G4_1(UGHOST1, 2);
Sprite G4_2(UGHOST2, 4);
Sprite G5_1(UGHOST1, 2);
Sprite G5_2(UGHOST2, 4);
Sprite G6_1(UGHOST1, 2);
Sprite G6_2(UGHOST2, 4);

 erase_message_window();

// set sprite member function pointer to MoveImage

SMoveImagePtr= &Sprite::MoveImage;
SEraseImagePtr= &Sprite::EraseImage;

// set sprite pointers to each sprite instance

UT1Ptr= &UT1;
UT2Ptr= &UT2;
DT1Ptr= &DT1;
DT2Ptr= &DT2;
RT1Ptr= &RT1;
RT2Ptr= &RT2;
LT1Ptr= &LT1;
LT2Ptr= &LT2;
G_1Ptr= &G_1;
G_2Ptr= &G_2;
G2_1Ptr= &G2_1;
G2_2Ptr= &G2_2;
G3_1Ptr= &G3_1;
G3_2Ptr= &G3_2;
G4_1Ptr= &G4_1;
G4_2Ptr= &G4_2;
```

Figure 8-1 *Continued.*

```
G5_1Ptr= &G5_1;
G5_2Ptr= &G5_2;
G6_1Ptr= &G6_1;
G6_2Ptr= &G6_2;

 init_game();

for(;;) {

   if(peep_flag) {
      Snd.On(500);
      }

   key= K1.Status();
   switch(key) {
      case SPACE:
         if(!missile_launched) {
            missiles_used+= 1;
            print_missiles_used();
            set_missile_position(param_case);
            missile_launched= 1;
            }
         break;
      case LEFT_ARROW:
         param= 1;
         break;
   case RIGHT_ARROW:
         param= 2;
         break;
   case DOWN_ARROW:
         param= 3;
         break;
   case UP_ARROW:
         param= 4;
         break;
   case ENTER:
         param= 0;
         break;
   case F10:
         goto exit_loop;
   }

   // update the position of the ghost

   ghost1_move();
   ghost2_move();
   ghost3_move();
   ghost4_move();
   ghost5_move();
   delay(delay_value);
```

```
if(peep_flag) {
   peep_flag= 0;
   Snd.Off();
   }

// check to see if the tank sprite collides with
// the ghost

switch(active_tank) {
case U_T:
  if(G_1.SpriteCollision(UT1.GetInfo(SInfoPtr))) {
      explosion();
      if(!game_over()) {
         init_game();
         }
      else {
         goto exit_loop;
         }
      }
  if(G2_1.SpriteCollision(UT1.GetInfo(SInfoPtr))) {
      explosion();
      if(!game_over()) {
         init_game();
         }
      else {
         goto exit_loop;
         }
    }
if(G3_1.SpriteCollision(UT1.GetInfo(SInfoPtr))) {
    explosion();
    if(!game_over()) {
       init_game();
       }
    else {
       goto exit_loop;
       }
    }
if(G4_1.SpriteCollision(UT1.GetInfo(SInfoPtr))) {
    explosion();
    if(!game_over()) {
       init_game();
       }
    else {
       goto exit_loop;
       }
    }
if(G5_1.SpriteCollision(UT1.GetInfo(SInfoPtr))) {
    explosion();
    if(!game_over()) {
       init_game();
       }
    else {
       goto exit_loop;
```

Figure 8-1 *Continued.*

```
                    }
                  }
                break;
            case D_T:
              if(G_1.SpriteCollision(DT1.GetInfo(SInfoPtr))) {
                  explosion();
                  if(!game_over()) {
                     init_game();
                     }
                  else {
                     goto exit_loop;
                     }
                  }
              if(G2_1.SpriteCollision(DT1.GetInfo(SInfoPtr))) {
                  explosion();
                  if(!game_over()) {
                     init_game();
                     }
                  else {
                     goto exit_loop;
                     }
                  }
              if(G3_1.SpriteCollision(DT1.GetInfo(SInfoPtr))) {
                  explosion();
                  if(!game_over()) {
                     init_game();
                     }
                  else {
                     goto exit_loop;
                     }
                  }
              if(G4_1.SpriteCollision(DT1.GetInfo(SInfoPtr))) {
                  explosion();
                  if(!game_over()) {
                     init_game();
                     }
                  else {
                     goto exit_loop;
                     }
                  }
              if(G5_1.SpriteCollision(DT1.GetInfo(SInfoPtr))) {
                  explosion();
                  if(!game_over()) {
                     init_game();
                     }
                  else {
                     goto exit_loop;
                     }
                  }
                break;
            case L_T:
```

```
if(G_1.SpriteCollision(LT1.GetInfo(SInfoPtr))) {
    explosion();
    if(!game_over()) {
       init_game();
       }
    else {
       goto exit_loop;
       }
    }
if(G2_1.SpriteCollision(LT1.GetInfo(SInfoPtr))) {
    explosion();
    if(!game_over()) {
       init_game();
       }
    else {
       goto exit_loop;
       }
    }
if(G3_1.SpriteCollision(LT1.GetInfo(SInfoPtr))) {
    explosion();
    if(!game_over()) {
       init_game();
       }
    else {
       goto exit_loop;
       }
    }
if(G4_1.SpriteCollision(LT1.GetInfo(SInfoPtr))) {
    explosion();
    if(!game_over()) {
       init_game();
       }
    else {
       goto exit_loop;
       }
    }
if(G5_1.SpriteCollision(LT1.GetInfo(SInfoPtr))) {
    explosion();
    if(!game_over()) {
       init_game();
       }
    else {
       goto exit_loop;
       }
    }
  break;
case R_T:
  if(G_1.SpriteCollision(RT1.GetInfo(SInfoPtr))) {
      explosion();
      if(!game_over()) {
         init_game();
         }
      else {
```

Figure 8-1 *Continued.*

```
                goto exit_loop;
                }
            }
        if(G2_1.SpriteCollision(RT1.GetInfo(SInfoPtr))) {
            explosion();
            if(!game_over()) {
                init_game();
                }
            else {
                goto exit_loop;
                }
            }
        if(G3_1.SpriteCollision(RT1.GetInfo(SInfoPtr))) {
            explosion();
            if(!game_over()) {
                init_game();
                }
            else {
                goto exit_loop;
                }
            }
        if(G4_1.SpriteCollision(RT1.GetInfo(SInfoPtr))) {
            explosion();
            if(!game_over()) {
                init_game();
                }
            else {
                goto exit_loop;
                }
            }
         if(G5_1.SpriteCollision(RT1.GetInfo(SInfoPtr))) {
            explosion();
            if(!game_over()) {
                init_game();
                }
            else {
                goto exit_loop;
                }
            }
          break;
        }

        // The missile travels at twice the speed of the
        // tank and ghost
        //
        // if the missile has been launched, then update its
        // position

        if(missile_launched) {
            launch_missile(missile_direction);
            launch_missile(missile_direction);
```

```
if(G_1.MissileCollision(M1.GetInfo(MInfoPtr))) {
   crash1();
  M1.EraseImage();
  G_1.EraseImage();
  G_2.EraseImage();
   launch_ghost1(12, 6);
   score+= 1;
   set_level();
   print_score(1);
   missile_launched= 0;
   }
if(G2_1.MissileCollision(M1.GetInfo(MInfoPtr))) {
   crash1();
  M1.EraseImage();
  G2_1.EraseImage();
  G2_2.EraseImage();
   g2_col= 12;
   g2_row= 6;
   ghost2_direction= 2;
   launch_ghost2(g2_row, g2_col);
   score+= 1;
   set_level();
   print_score(1);
   missile_launched= 0;
   }
if(G3_1.MissileCollision(M1.GetInfo(MInfoPtr))) {
   crash1();
  M1.EraseImage();
  G3_1.EraseImage();
  G3_2.EraseImage();
   g3_col= 582;
   g3_row= 424;
   ghost3_direction= 3;
   launch_ghost3(g3_row, g3_col);
   score+= 1;
   set_level();
   print_score(1);
   missile_launched= 0;
   }
if(G4_1.MissileCollision(M1.GetInfo(MInfoPtr))) {
   crash1();
  M1.EraseImage();
G4_1.EraseImage();
G4_2.EraseImage();
 g4_col= 582;
 g4_row= 424;
 ghost4_direction= 3;
 launch_ghost4(g4_row, g4_col);
 score+= 1;
 set_level();
 print_score(1);
 missile_launched= 0;
 }
```

Figure 8-1 *Continued.*

```
if(G5_1.MissileCollision(M1.GetInfo(MInfoPtr))) {
   crash1();
  M1.EraseImage();
  G5_1.EraseImage();
  G5_2.EraseImage();
   g5_col= 582;
   g5_row= 424;
      ghost5_direction= 3;
      launch_ghost5(g5_row, g5_col);
      score+= 1;
      set_level();
      print_score(1);
      missile_launched= 0;
      }
   }

// evaluate key press to determine if there has been
// a legal tank move

if(legal_sprite_move(param)) {
   poll_sprite(param);
   param_case= param;
   set_active_tank(param);
   }

      // otherwise continue moving sprite until boundary
      // is reached

      else {
         if(legal_sprite_move(param_case)) {
            poll_sprite(param_case);
            }
         }

      // if the missile has been launched, then update its
      // position

      if(missile_launched) {
/*************************************************
         launch_missile(missile_direction, &missile1);
         launch_missile(missile_direction, &missile1);
      if(sprite_missile_collision(&ughost_1, &missile1)) {
         crash1();
         erase_missile_image(&missile1);
         erase_sprite_image(&ughost_1);
         erase_sprite_image(&ughost_2);
         launch_ghost1(12, 6);
         score+= 1;
         print_score(1);
         set_level();
         missile_launched= 0;
         }
```

```
if(sprite_missile_collision(&ughost2_1, &missile1)) {
   crash1();
   erase_missile_image(&missile1);
   erase_sprite_image(&ughost2_1);
   erase_sprite_image(&ughost2_2);
   g2_col= 12;
   g2_row= 6;
   ghost2_direction= 2;
   launch_ghost2(g2_row, g2_col);
   score+= 1;
   print_score(1);
   set_level();
   missile_launched= 0;
   }
if(sprite_missile_collision(&ughost3_1, &missile1)) {
   crash1();
   erase_missile_image(&missile1);
   erase_sprite_image(&ughost3_1);
   erase_sprite_image(&ughost3_2);
   g3_col= 582;
   g3_row= 424;
   ghost3_direction= 3;
   launch_ghost3(g3_row, g3_col);
   score+= 1;
   print_score(1);
   set_level();
   missile_launched= 0;
   }
if(sprite_missile_collision(&ughost4_1, &missile1)) {
   crash1();
   erase_missile_image(&missile1);
   erase_sprite_image(&ughost4_1);
   erase_sprite_image(&ughost4_2);
   g4_col= 582;
   g4_row= 424;
   ghost4_direction= 3;
   launch_ghost4(g4_row, g4_col);
   score+= 1;
   print_score(1);
   set_level();
   missile_launched= 0;
   }
if(sprite_missile_collision(&ughost5_1, &missile1)) {
   crash1();
   erase_missile_image(&missile1);
   erase_sprite_image(&ughost5_1);
   erase_sprite_image(&ughost5_2);
   g5_col= 582;
   g5_row= 424;
   ghost5_direction= 3;
   launch_ghost5(g5_row, g5_col);
   score+= 1;
   print_score(1);
```

Figure 8-1 *Continued.*

```
                set_level();
                missile_launched= 0;
                }
*****************************************************************/
            }
        }

exit_loop:
  P1.Destroy();

   closegraph();

}

void set_missile_position(int param)
{
   crash1();
   switch(param) {
      // left
      case  1:
         missile_col= col - 7;
         missile_row= row + SPRITE_TRACK_OFFSET - 4;
  //          move_missile_image(missile_col, missile_row, &missile1);
         missile_direction= param;
         missile_launched= 1;
         break;
      // right
      case  2:
         missile_col= col + SPRITE_TRACK  - 4;
         missile_row= row + SPRITE_TRACK_OFFSET - 4;
//          move_missile_image(missile_col, missile_row, &missile1);
         missile_direction= param;
         missile_launched= 1;
         break;
      // down
      case  3:
         missile_col= col + SPRITE_TRACK_OFFSET - 6;
         missile_row= row + SPRITE_TRACK - 4;
//          move_missile_image(missile_col, missile_row, &missile1);
         missile_direction= param;
         missile_launched= 1;
         break;
      // up
      case  4:
         missile_col= col + SPRITE_TRACK_OFFSET - 6;
         missile_row= row - 6;
//          move_missile_image(missile_col, missile_row, &missile1);
         missile_direction= param;
         missile_launched= 1;
         break;
      }
      (M1Ptr->*MMoveImagePtr)(missile_col, missile_row);
```

```
//    onSound(100);
}

void launch_missile(int param)
{

   switch(param) {
      // left missile motion
      case 1:
         if(missile_col < 16) {
            missile_launched= 0;
           (M1Ptr->*MEraseImagePtr)();
//            erase_missile_image(mi);
            peep();
            print_score(1);
            return;
            }
        (M1Ptr->*MMoveImagePtr)(missile_col--, missile_row);
        (M1Ptr->*MMoveImagePtr)(missile_col--, missile_row);
        (M1Ptr->*MMoveImagePtr)(missile_col--, missile_row);
        (M1Ptr->*MMoveImagePtr)(missile_col--, missile_row);
        (M1Ptr->*MMoveImagePtr)(missile_col--, missile_row);
        (M1Ptr->*MMoveImagePtr)(missile_col--, missile_row);
        (M1Ptr->*MMoveImagePtr)(missile_col--, missile_row);
        (M1Ptr->*MMoveImagePtr)(missile_col--, missile_row);
/*********************************
            move_missile_image(missile_col--, missile_row, &missile1);
            move_missile_image(missile_col--, missile_row, &missile1);
            move_missile_image(missile_col--, missile_row, &missile1);
            move_missile_image(missile_col--, missile_row, &missile1);
            move_missile_image(missile_col--, missile_row, &missile1);
            move_missile_image(missile_col--, missile_row, &missile1);
            move_missile_image(missile_col--, missile_row, &missile1);
            move_missile_image(missile_col--, missile_row, &missile1);
            onSound(missile_col);
*********************************/
            break;
         // right missile motion
         case 2:
            if(missile_col > max_x - 16) {
               missile_launched= 0;
              (M1Ptr->*MEraseImagePtr)();
//               erase_missile_image(mi);
              peep();
              print_score(1);
              return;
              }
           (M1Ptr->*MMoveImagePtr)(missile_col++, missile_row);
           (M1Ptr->*MMoveImagePtr)(missile_col++, missile_row);
           (M1Ptr->*MMoveImagePtr)(missile_col++, missile_row);
           (M1Ptr->*MMoveImagePtr)(missile_col++, missile_row);
           (M1Ptr->*MMoveImagePtr)(missile_col++, missile_row);
           (M1Ptr->*MMoveImagePtr)(missile_col++, missile_row);
```

Figure 8-1 *Continued.*

```
            (M1Ptr->*MMoveImagePtr)(missile_col++, missile_row);
            (M1Ptr->*MMoveImagePtr)(missile_col++, missile_row);
/*************************************************
               move_missile_image(missile_col++, missile_row, &missile1);
               move_missile_image(missile_col++, missile_row, &missile1);
               move_missile_image(missile_col++, missile_row, &missile1);
               move_missile_image(missile_col++, missile_row, &missile1);
               move_missile_image(missile_col++, missile_row, &missile1);
               move_missile_image(missile_col++, missile_row, &missile1);
               onSound(missile_col);
*************************************************/
               break;
            // down missile motion
            case 3:
               if(missile_row > max_y - 16) {
                  missile_launched= 0;
                  (M1Ptr->*MEraseImagePtr)();
//                   erase_missile_image(mi);
                  peep();
                  print_score(1);
                  return;
                  }
               (M1Ptr->*MMoveImagePtr)(missile_col, missile_row++);
               (M1Ptr->*MMoveImagePtr)(missile_col, missile_row++);
               (M1Ptr->*MMoveImagePtr)(missile_col, missile_row++);
               (M1Ptr->*MMoveImagePtr)(missile_col, missile_row++);
               (M1Ptr->*MMoveImagePtr)(missile_col, missile_row++);
               (M1Ptr->*MMoveImagePtr)(missile_col, missile_row++);
               (M1Ptr->*MMoveImagePtr)(missile_col, missile_row++);
               (M1Ptr->*MMoveImagePtr)(missile_col, missile_row++);
/*************************************
               move_missile_image(missile_col, missile_row++, &missile1);
               move_missile_image(missile_col, missile_row++, &missile1);
               move_missile_image(missile_col, missile_row++, &missile1);
               move_missile_image(missile_col, missile_row++, &missile1);
               move_missile_image(missile_col, missile_row++, &missile1);
               move_missile_image(missile_col, missile_row++, &missile1);
               move_missile_image(missile_col, missile_row++, &missile1);
               move_missile_image(missile_col, missile_row++, &missile1);
               onSound(missile_row);
*************************************************/
               break;
            // up missile motion
            case 4:
               if(missile_row < 16) {
                  missile_launched= 0;
                  (M1Ptr->*MEraseImagePtr)();
//                   erase_missile_image(mi);
                  peep();
                  print_score(1);
```

```
        return;
        }
(M1Ptr->*MMoveImagePtr)(missile_col, missile_row--);
(M1Ptr->*MMoveImagePtr)(missile_col, missile_row--);
(M1Ptr->*MMoveImagePtr)(missile_col, missile_row--);
(M1Ptr->*MMoveImagePtr)(missile_col, missile_row--);
(M1Ptr->*MMoveImagePtr)(missile_col, missile_row--);
(M1Ptr->*MMoveImagePtr)(missile_col, missile_row--);
(M1Ptr->*MMoveImagePtr)(missile_col, missile_row--);
      (M1Ptr->*MMoveImagePtr)(missile_col, missile_row--);
/*************************************
          move_missile_image(missile_col, missile_row--, &missile1);
          move_missile_image(missile_col, missile_row--, &missile1);
          move_missile_image(missile_col, missile_row--, &missile1);
          move_missile_image(missile_col, missile_row--, &missile1);
          move_missile_image(missile_col, missile_row--, &missile1);
          move_missile_image(missile_col, missile_row--, &missile1);
          move_missile_image(missile_col, missile_row--, &missile1);
          move_missile_image(missile_col, missile_row--, &missile1);
          onSound(missile_row);
*********************************************/
          break;
       }

}

void poll_sprite(int param)
{

   switch(param) {
      // left tank motion
      case 1:
        (LT1Ptr->*SMoveImagePtr)(col--, row);
        (LT2Ptr->*SMoveImagePtr)(col, row);
//          move_sprite_image(col--, row, &tank_l1);
//          move_sprite_image(col, row, &tank_l2);
         break;

      // right tank motion
      case 2:
        (RT1Ptr->*SMoveImagePtr)(col++, row);
        (RT2Ptr->*SMoveImagePtr)(col, row);
//          move_sprite_image(col++, row, &tank_r1);
//          move_sprite_image(col, row, &tank_r2);
         break;
      // down tank motion
      case 3:
        (DT1Ptr->*SMoveImagePtr)(col, row++);
        (DT2Ptr->*SMoveImagePtr)(col, row);
//          move_sprite_image(col, row++, &tank_d1);
//          move_sprite_image(col, row, &tank_d2);
         break;
      // up tank motion
```

Figure 8-1 *Continued.*

```
        case 4:
          (UT1Ptr->*SMoveImagePtr)(col, row--);
          (UT2Ptr->*SMoveImagePtr)(col, row);
//          move_sprite_image(col, row--, &tank_u1);
//          move_sprite_image(col, row, &tank_u2);
           break;
        }
}

int legal_sprite_move(int param)
{

   switch(param) {
      // left tank motion
      case 1:
         if((col > 12) && ((row == 6) ¦¦
                           (row == 424) ¦¦
                           (row == midy - SPRITE_TRACK_OFFSET + 4))) {
            return 1;
            }
         break;

      // right tank motion
      case 2:
         if((col < 582) && ((row == 6)   ¦¦
                            (row == 424) ¦¦
                            (row == midy - SPRITE_TRACK_OFFSET + 4))) {
            return 1;
            }
         break;
      // down tank motion
      case 3:
         if((row < 424) && ((col == 12)  ¦¦
                            (col == 582)¦¦
                            (col == midx - SPRITE_TRACK_OFFSET + 8))) {
            return 1;
            }
         break;
      // up tank motion
         case 4:
            if((row > 6)  && ((col == 12)   ¦¦
                              (col == 582) ¦¦
                              (col == midx - SPRITE_TRACK_OFFSET + 8))) {
               return 1;
               }
            break;
         }
      return 0;
   }
```

```
void launch_ghost1(int r, int c)
{
   g1_col= c;
   g1_row= r;
  (G_1Ptr->*SMoveImagePtr)(c, r);
  (G_2Ptr->*SMoveImagePtr)(c, r);
//   move_sprite_image(c, r, &ughost_1);
//   move_sprite_image(c, r, &ughost_2);
}

void ghost1_move()
{

   // track ghost 1 to move right in center row

   if(difficulty_level > LEVEL_1) {
      if((g1_row == 6)   ||
         (g1_row == 424) ||
         (g1_row == midy - SPRITE_TRACK_OFFSET + 4)) {
         if(g1_row == row) {
            if(g1_col > col) {
               ghost1_direction= 3;
               goto ghost1_direction_intercept;
               }
            if(g1_col < col) {
               ghost1_direction= 1;
               goto ghost1_direction_intercept;
               }
         }
      }
   }

if(difficulty_level > LEVEL_2) {

   // move ghost down from row 1 if tank is on center row
   // or bottom row

   if(g1_row < row) {
      if((g1_col == 12)   ||
         (g1_col == 582)  ||
         (g1_col == midx - SPRITE_TRACK_OFFSET + 8)) {
         if(row >= midy - SPRITE_TRACK_OFFSET + 4) {
            ghost1_direction= 2;
            goto ghost1_direction_intercept;
            }
         }
      }
   }

if(difficulty_level > LEVEL_3) {

   // move ghost up from row 424 if tank is on or above
   // center
```

Figure 8-1 *Continued.*

```
if(g1_row > row) {
   if((g1_col == 12)    ||
      (g1_col == 582)   ||
      (g1_col == midx - SPRITE_TRACK_OFFSET + 8)) {
      if(row <= midy - SPRITE_TRACK_OFFSET + 4) {
         ghost1_direction= 4;
         goto ghost1_direction_intercept;
            }
         }
      }
   }

// bypass other conditions to avoid unnecessary
// conditional checks

ghost1_direction_intercept:

// make ghost1 move

switch(ghost1_direction) {
   // right motion
   case 1:
      if(g1_col < 582) {
          (G_1Ptr->*SMoveImagePtr)(g1_col++, g1_row);
          (G_2Ptr->*SMoveImagePtr)(g1_col, g1_row);
//           move_sprite_image(g1_col++, g1_row, &ughost_1);
//           move_sprite_image(g1_col, g1_row, &ughost_2);
         }
      else {
         ghost1_direction= 2;
         }
      break;

   // down motion

   case 2:
      if(g1_row < 424) {
         if(difficulty_level >= LEVEL_3) {
             (G_1Ptr->*SMoveImagePtr)(g1_col, g1_row++);
             (G_2Ptr->*SMoveImagePtr)(g1_col, g1_row);
//            move_sprite_image(g1_col, g1_row++, &ughost_1);
//            move_sprite_image(g1_col, g1_row, &ughost_2);
           }
          (G_1Ptr->*SMoveImagePtr)(g1_col, g1_row++);
          (G_2Ptr->*SMoveImagePtr)(g1_col, g1_row);
//           move_sprite_image(g1_col, g1_row++, &ughost_1);
//           move_sprite_image(g1_col, g1_row, &ughost_2);
         }
      else {
         ghost1_direction= 3;
         }
      break;
```

```
        // left motion
        case 3:
           if(g1_col > 12) {
               (G_1Ptr->*SMoveImagePtr)(g1_col--, g1_row);
               (G_2Ptr->*SMoveImagePtr)(g1_col, g1_row);
//              move_sprite_image(g1_col--, g1_row, &ughost_1);
//              move_sprite_image(g1_col, g1_row, &ughost_2);
             }
           else {
              ghost1_direction= 4;
              }
           break;

        // up motion
        case 4:
           if(g1_row > 6) {
               (G_1Ptr->*SMoveImagePtr)(g1_col, g1_row--);
               (G_2Ptr->*SMoveImagePtr)(g1_col, g1_row);
//              move_sprite_image(g1_col, g1_row--, &ughost_1);
//              move_sprite_image(g1_col, g1_row, &ughost_2);
             }
           else {
              ghost1_direction= 1;
              }
           break;
        }
}

void launch_ghost2(int r, int c)
{
   g2_col= c;
   g2_row= r;
  (G2_1Ptr->*SMoveImagePtr)(c, r);
  (G2_2Ptr->*SMoveImagePtr)(c, r);
//   move_sprite_image(c, r, &ughost2_1);
//   move_sprite_image(c, r, &ughost2_2);
}

void ghost2_move()
{
// track ghost 2 to move right in center row

if(difficulty_level > LEVEL_5) {
   if((g2_row == 6)   ||
      (g2_row == 424) ||
      (g2_row == midy - SPRITE_TRACK_OFFSET + 4)) {
      if(g2_row == row) {
         if(g2_col > col) {
            ghost2_direction= 4;
            goto ghost2_direction_intercept;
            }
         if(g2_col < col) {
```

Figure 8-1 *Continued.*

```
                        ghost2_direction= 2;
                        goto ghost2_direction_intercept;
                        }
                    }
                }
            }

            if(difficulty_level > LEVEL_6) {

               // move ghost down from row 1 if tank is on center row
               // or bottom row

               if(g2_row < row) {
                  if((g2_col == 12)   ||
                     (g2_col == 582)  ||
                     (g2_col == midx - SPRITE_TRACK_OFFSET + 8)) {
                     if(row >= midy - SPRITE_TRACK_OFFSET + 4) {
                        ghost2_direction= 1;
                        goto ghost2_direction_intercept;
                        }
                     }
                  }
               }

            if(difficulty_level > LEVEL_7) {

               // move ghost up from row 424 if tank is on or above
               // center row

               if(g2_row > row) {
                  if((g2_col == 12)   ||
                     (g2_col == 582)  ||
                     (g2_col == midx - SPRITE_TRACK_OFFSET + 8)) {
                     if(row <= midy - SPRITE_TRACK_OFFSET + 4) {
                        ghost2_direction= 3;
                        goto ghost2_direction_intercept;
                        }
                     }
                  }
               }

            // bypass other conditions to avoid unnecessary
            // conditional checks

            ghost2_direction_intercept:

            switch(ghost2_direction) {
               // right motion
               case 2:
                  if(g2_col < 582) {
                     (G2_1Ptr->*SMoveImagePtr)(g2_col++, g2_row);
                     (G2_2Ptr->*SMoveImagePtr)(g2_col, g2_row);
```

```
//              move_sprite_image(g2_col++, g2_row, &ughost2_1);
//              move_sprite_image(g2_col, g2_row, &ughost2_2);
            }
         else {
            ghost2_direction= 3;
            }
         break;

      // down motion

      case 1:
         if(g2_row < 424) {
             (G2_1Ptr->*SMoveImagePtr)(g2_col, g2_row++);
             (G2_2Ptr->*SMoveImagePtr)(g2_col, g2_row);
//            move_sprite_image(g2_col, g2_row++, &ughost2_1);
//            move_sprite_image(g2_col, g2_row, &ughost2_2);
            }
         else {
            ghost2_direction= 2;
            }
         break;

      // left motion
      case 4:
         if(g2_col > 12) {
             (G2_1Ptr->*SMoveImagePtr)(g2_col--, g2_row);
             (G2_2Ptr->*SMoveImagePtr)(g2_col, g2_row);
//            move_sprite_image(g2_col--, g2_row, &ughost2_1);
//            move_sprite_image(g2_col, g2_row, &ughost2_2);
            }
         else {
            ghost2_direction= 1;
            }
         break;

      // up motion
      case 3:
         if(g2_row > 6) {
             (G2_1Ptr->*SMoveImagePtr)(g2_col, g2_row--);
             (G2_2Ptr->*SMoveImagePtr)(g2_col, g2_row);
//            move_sprite_image(g2_col, g2_row--, &ughost2_1);
//            move_sprite_image(g2_col, g2_row, &ughost2_2);
            }
         else {
            ghost2_direction= 4;
            }
         break;
      }
}

void launch_ghost3(int r, int c)
{
   g3_col= c;
```

Figure 8-1 *Continued.*

```
   g3_row= r;
  (G3_1Ptr->*SMoveImagePtr)(c, r);
  (G3_2Ptr->*SMoveImagePtr)(c, r);
//   move_sprite_image(c, r, &ughost3_1);
//   move_sprite_image(c, r, &ughost3_2);
}

void ghost3_move()
{

   // track ghost 3 to move right in center row

   if(difficulty_level > LEVEL_5) {
      if((g3_row == 6)   ||
         (g3_row == 424) ||
         (g3_row == midy - SPRITE_TRACK_OFFSET + 4)) {
         if(g3_row == row) {
            if(g3_col > col) {
               ghost3_direction= 4;
               goto ghost3_direction_intercept;
               }
            if(g3_col < col) {
               ghost3_direction= 2;
               goto ghost3_direction_intercept;
               }
         }
      }
   }

if(difficulty_level > LEVEL_3) {

   // move ghost down from row 1 if tank is on center row
   // or bottom row

   if(g3_row < row) {
      if((g3_col == 12)   ||
         (g3_col == 582)  ||
         (g3_col == midx - SPRITE_TRACK_OFFSET + 8)) {
         if(row >= midy - SPRITE_TRACK_OFFSET + 4) {
            ghost3_direction= 1;
            goto ghost3_direction_intercept;
            }
         }
      }
   }

if(difficulty_level > LEVEL_1) {

   // move ghost up from row 424 if tank is on or above
   // center row

   if(g3_row > row) {
```

```
if((g3_col == 12)   ||
   (g3_col == 582)  ||
   (g3_col == midx - SPRITE_TRACK_OFFSET + 8)) {
   if(row <= midy - SPRITE_TRACK_OFFSET + 4) {
      ghost3_direction= 3;
      goto ghost3_direction_intercept;
      }
   }

   // bypass other conditions to avoid unnecessary
   // conditional checks

   ghost3_direction_intercept:

   switch(ghost3_direction) {
      // right motion
      case 2:
         if(g3_col < 582) {
             (G3_1Ptr->*SMoveImagePtr)(g3_col++, g3_row);
             (G3_2Ptr->*SMoveImagePtr)(g3_col, g3_row);
//            move_sprite_image(g3_col++, g3_row, &ughost3_1);
//            move_sprite_image(g3_col, g3_row, &ughost3_2);
           }
         else {
            ghost3_direction= 4;
            }
         break;

      // down motion

      case 1:
         if(g3_row < center_sprite_y) {
             (G3_1Ptr->*SMoveImagePtr)(g3_col, g3_row++);
             (G3_2Ptr->*SMoveImagePtr)(g3_col, g3_row);
//            move_sprite_image(g3_col, g3_row++, &ughost3_1);
//            move_sprite_image(g3_col, g3_row, &ughost3_2);
           }
         else {
            ghost3_direction= 2;
            }
         break;

      // left motion
      case 4:
         if(g3_col > 12) {
            if(difficulty_level >= LEVEL_3) {
             (G3_1Ptr->*SMoveImagePtr)(g3_col--, g3_row);
             (G3_2Ptr->*SMoveImagePtr)(g3_col, g3_row);
//              move_sprite_image(g3_col--, g3_row, &ughost3_1);
//              move_sprite_image(g3_col, g3_row, &ughost3_2);
             }
```

Figure 8-1 *Continued.*

```
                (G3_1Ptr->*SMoveImagePtr)(g3_col--, g3_row);
                (G3_2Ptr->*SMoveImagePtr)(g3_col, g3_row);
//               move_sprite_image(g3_col--, g3_row, &ughost3_1);
//               move_sprite_image(g3_col, g3_row, &ughost3_2);
             }
          else {
             ghost3_direction= 1;
             }
          break;

       // up motion
       case 3:
          if(g3_row > 6) {
              (G3_1Ptr->*SMoveImagePtr)(g3_col, g3_row--);
              (G3_2Ptr->*SMoveImagePtr)(g3_col, g3_row);
//             move_sprite_image(g3_col, g3_row--, &ughost3_1);
//             move_sprite_image(g3_col, g3_row, &ughost3_2);
             }
          else {
             ghost3_direction= 4;
             }
          break;
       }
}

void launch_ghost4(int r, int c)
{
   g4_col= c;
   g4_row= r;
  (G4_1Ptr->*SMoveImagePtr)(c, r);
  (G4_2Ptr->*SMoveImagePtr)(c, r);
//   move_sprite_image(c, r, &ughost4_1);
//   move_sprite_image(c, r, &ughost4_2);
}

void ghost4_move()
{
   // track ghost 4 to move right in center row

   if(difficulty_level >= LEVEL_1) {
   if((g4_row == 6)   ||
      (g4_row == 424) ||
      (g4_row == midy - SPRITE_TRACK_OFFSET + 4)) {
      if(g4_row == row) {
         if(g4_col > col) {
            ghost4_direction= 4;
            goto ghost4_direction_intercept;
            }
         if(g4_col < col) {
            ghost4_direction= 3;
            goto ghost4_direction_intercept;
            }
```

```
            }
         }
      }

   if(difficulty_level > LEVEL_2) {

      // move ghost down from row 1 if tank is on center row
      // or bottom row
      if(g4_row < row) {
         if((g4_col == 12)   ||
            (g4_col == 582)  ||
            (g4_col == midx - SPRITE_TRACK_OFFSET + 8)) {
            if(row >= midy - SPRITE_TRACK_OFFSET + 4) {
               ghost4_direction= 1;
               goto ghost4_direction_intercept;
               }
            }
         }
      }

   if(difficulty_level > LEVEL_3) {

      // move ghost up from row 424 if tank is on or above
      // center row
      if(g4_row > row) {
         if((g4_col == 12)   ||
            (g4_col == 582)  ||
            (g4_col == midx - SPRITE_TRACK_OFFSET + 8)) {
            if(row <= midy - SPRITE_TRACK_OFFSET + 4) {
               ghost4_direction= 3;
               goto ghost4_direction_intercept;
               }
            }
         }
      }

   // bypass other conditions to avoid unnecessary
   // conditional checks

   ghost4_direction_intercept:

      switch(ghost4_direction) {
         // right motion
         case 2:
            if(g4_col < 582) {
               (G4_1Ptr->*SMoveImagePtr)(g4_col++, g4_row);
               (G4_2Ptr->*SMoveImagePtr)(g4_col, g4_row);
//              move_sprite_image(g4_col++, g4_row, &ughost4_1);
//              move_sprite_image(g4_col, g4_row, &ughost4_2);
              }
            else {
```

Figure 8-1 *Continued.*

```
                ghost4_direction= 3;
                }
            break;

        // down motion

        case 1:
            if(g4_row < 424) {
                (G4_1Ptr->*SMoveImagePtr)(g4_col, g4_row++);
                (G4_2Ptr->*SMoveImagePtr)(g4_col, g4_row);
//               move_sprite_image(g4_col, g4_row++, &ughost4_1);
//               move_sprite_image(g4_col, g4_row, &ughost4_2);
              }
            else {
               ghost4_direction= 2;
               }
            break;

        // left motion
        case 4:
            if(g4_col > 12) {
                (G4_1Ptr->*SMoveImagePtr)(g4_col--, g4_row);
                (G4_2Ptr->*SMoveImagePtr)(g4_col, g4_row);
//               move_sprite_image(g4_col--, g4_row, &ughost4_1);
//               move_sprite_image(g4_col, g4_row, &ughost4_2);
              }
            else {
               ghost4_direction= 1;
               }
            break;

        // up motion
        case 3:
            if(g4_row > 6) {
                (G4_1Ptr->*SMoveImagePtr)(g4_col, g4_row--);
                (G4_2Ptr->*SMoveImagePtr)(g4_col, g4_row);
//               move_sprite_image(g4_col, g4_row--, &ughost4_1);
//               move_sprite_image(g4_col, g4_row, &ughost4_2);
              }
            else {
                ghost4_direction= 4;
                }
            break;
        }

}

void launch_ghost5(int r, int c)
{
   g5_col= c;
   g5_row= r;
  (G5_1Ptr->*SMoveImagePtr)(c, r);
```

```
   (G5_2Ptr->*SMoveImagePtr)(c, r);
//    move_sprite_image(c, r, &ughost5_1);
//    move_sprite_image(c, r, &ughost5_2);
}

void ghost5_move()
{

    switch(ghost5_direction) {
        // right motion
        case 1:
            if(g5_col < 582) {
                (G5_1Ptr->*SMoveImagePtr)(g5_col++, g5_row);
                (G5_2Ptr->*SMoveImagePtr)(g5_col, g5_row);
//                move_sprite_image(g5_col++, g5_row, &ughost5_1);
//                move_sprite_image(g5_col, g5_row, &ughost5_2);
               }
            else {
               ghost5_direction= 2;
               }
            break;

        // down motion

        case 2:
            if(g5_row < 424) {
                (G5_1Ptr->*SMoveImagePtr)(g5_col, g5_row++);
                (G5_2Ptr->*SMoveImagePtr)(g5_col, g5_row);
//                move_sprite_image(g5_col, g5_row++, &ughost5_1);
//                move_sprite_image(g5_col, g5_row, &ughost5_2);
               }
            else {
               ghost5_direction= 3;
               }
            break;

        // left motion
        case 3:
            if(g5_col > 12) {
                (G5_1Ptr->*SMoveImagePtr)(g5_col--, g5_row);
                (G5_2Ptr->*SMoveImagePtr)(g5_col, g5_row);
//                move_sprite_image(g5_col--, g5_row, &ughost5_1);
//                move_sprite_image(g5_col, g5_row, &ughost5_2);
               }
            else {
               ghost5_direction= 4;
               }
            break;

        // up motion
        case 4:
            if(g5_row > 6) {
                (G5_1Ptr->*SMoveImagePtr)(g5_col, g5_row--);
```

Figure 8-1 *Continued.*

```
                (G5_2Ptr->*SMoveImagePtr)(g5_col, g5_row);
//               move_sprite_image(g5_col, g5_row--, &ughost5_1);
//               move_sprite_image(g5_col, g5_row, &ughost5_2);
            }
         else {
            ghost5_direction= 1;
            }
            break;
         }
}

void set_active_tank(int param)
{

   switch(param) {
      case 1:
         active_tank= L_T;
         break;
      case 2:
         active_tank= R_T;
         break;
      case 3:
         active_tank= D_T;
         break;
      case 4:
         active_tank= U_T;
         break;
      }
}

void crash()
{
  (SndPtr->*SoundOnPtr)(300);
  delay(2);
  (SndPtr->*SoundOffPtr)();
}

void crash1()
{
  (SndPtr->*SoundOnPtr)(1300);
  delay(4);
  (SndPtr->*SoundOffPtr)();
}

void peep()
{
   peep_flag= 1;
}

void print_missiles_used()
{
char score_buf[25];
```

```
int  cnt;

   memset(score_buf, 0, 25);
   setwritemode(COPY_PUT);
   setcolor(DARKGRAY);
   line(100 + (12 * 8),  116, 100 + (16 * 8), 116);
   for(cnt= 0; cnt < 10; cnt++) {
      line(100 + (12 * 8),  116 + cnt, 100 + (16 * 8), 116 + cnt);
      }
   moveto(100 + (12 * 8), 116);
   setcolor(WHITE);
   memset(score_buf, 0, 25);
   sprintf(score_buf,"%03d", missiles_used);
   outtext(score_buf);
}

void print_score(int hit)
{
char score_buf[25];
int  cnt;

   memset(score_buf, 0, 25);
   setwritemode(COPY_PUT);
   setcolor(DARKGRAY);
   line(100 + (12 * 8),  100, 100 + (16 * 8), 100);
   for(cnt= 0; cnt < 10; cnt++) {
      line(100 + (12 * 8),  100 + cnt, 100 + (16 * 8), 100 + cnt);
      }
moveto(100 + (12 * 8), 100);
setcolor(WHITE);
memset(score_buf, 0, 25);
sprintf(score_buf,"%04d", score);
outtext(score_buf);

if(hit) {
   total_score+= (100 * difficulty_level);
   adjusted_score= total_score - (50 * (missiles_used - score));
   }
else {
   adjusted_score= 0;
   }

memset(score_buf, 0, 25);
setwritemode(COPY_PUT);
setcolor(DARKGRAY);
line(100 + (12 * 8),  148, 100 + (16 * 8), 148);
for(cnt= 0; cnt < 10; cnt++) {
   line(100 + (12 * 8),  148 + cnt, 100 + (19 * 8), 148 + cnt);
   }
moveto(100 + (12 * 8), 148);
setcolor(WHITE);
memset(score_buf, 0, 25);
sprintf(score_buf,"%071d", adjusted_score);
```

Figure 8-1 *Continued.*

```
    outtext(score_buf);

}

void print_level()
{
int  cnt;

    setwritemode(COPY_PUT);
    setcolor(BLUE);
    line(100 + (12 * 8),  132, 100 + (15 * 8), 132);
    for(cnt= 0; cnt < 10; cnt++) {
       line(100 + (11 * 8),  132 + cnt, 100 + (15 * 8), 132 + cnt);
       }
    moveto(100 + (12 * 8), 132);
    setcolor(WHITE);
    switch(difficulty_level) {
       case LEVEL_1:
          outtext("1");
          break;
       case LEVEL_2:
          delay_value= DELAY7_MS;
          outtext("2");
          break;
    case LEVEL_3:
       delay_value= DELAY6_MS;
       outtext("3");
       break;
    case LEVEL_4:
       delay_value= DELAY5_MS;
       outtext("4");
       break;
    case LEVEL_5:
       delay_value= DELAY4_MS;
       outtext("5");
       break;
    case LEVEL_6:
       delay_value= DELAY3_MS;
       outtext("6");
       break;
    case LEVEL_7:
       delay_value= DELAY2_MS;
       outtext("7");
       break;
    case LEVEL_8:
       outtext("8");
       break;
    case LEVEL_9:
       outtext("9");
       break;
    }
```

```
}

void init_game()
{
   erase_message_window();
   score= 0;
   total_score= 0;
   adjusted_score= 0;
   missiles_used= 0;
   delay_value= DELAY8_MS;
   difficulty_level= LEVEL_1;
   ghost1_direction= 1;
   ghost2_direction= 1;
   ghost3_direction= 1;
   ghost4_direction= 1;
   ghost5_direction= 1;
   ghost1_launched= 0;
   ghost2_launched= 0;
   ghost3_launched= 0;
   ghost4_launched= 0;
   ghost5_launched= 0;

   missile_launched= 0;
   g1_col= 12;
   g1_row= 6;
   ghost1_direction= 1;

   g2_col= 12;
   g2_row= 6;
   ghost2_direction= 2;

   g3_col= 582;
   g3_row= 424;
   ghost3_direction= 3;

   g4_col= 582;
   g4_row= 424;
   ghost4_direction= 3;

   g5_col= 582;
   g5_row= 424;
   ghost5_direction= 3;

   col= center_sprite_x;
   row= center_sprite_y;

   col2= 12;
   row2= 100;

   moveto(100, 100);
   setcolor(WHITE);
   outtext("Ghosts Hit: ");
```

Figure 8-1 *Continued.*

```
   moveto(100, 116);
   setcolor(WHITE);
   outtext("Missiles: ");

   moveto(100, 132);
   setcolor(WHITE);
   outtext("Difficulty: ");

   moveto(100, 148);
   setcolor(WHITE);
   outtext("Your Score: ");

   print_score(0);

   print_missiles_used();

   print_level();

   (DT1Ptr->*SMoveImagePtr)(col, row);
   (DT2Ptr->*SMoveImagePtr)(col, row);
//   move_sprite_image(col, row, &tank_d2);
//   move_sprite_image(col, row, &tank_d1);
   active_tank= D_T;

   launch_ghost1(12, 6);
   launch_ghost2(12, 6);
   launch_ghost3(g3_row, g3_col);
   launch_ghost4(g4_row, g4_col);
}

int game_over()
{
char key;
   remove_sprites();

   moveto(100, 156);
   setcolor(RED);
   outtext("Game Over");
   moveto(100, 164);
   outtext("Play Again? (Y) (N)");

   moveto(100, 156);
   setcolor(RED);
   outtext("Game Over: ");
  key= (KeyPtr->*KGetPtr)();
//   key= (char)gtKey();

   if((key == 'Y') || (key == 'y')) {
      return 0;
      }
   else  {
```

```
        return 1;
        }

}

void set_level()
{
switch(score) {
   case 6:
      difficulty_level= LEVEL_2;
      break;
   case 12:
      difficulty_level= LEVEL_3;
      break;
   case 18:
      difficulty_level= LEVEL_4;
      break;
   case 24:
      difficulty_level= LEVEL_5;
      break;
   case 30:
      difficulty_level= LEVEL_6;
      break;
      case 36:
         difficulty_level= LEVEL_7;
         break;
      case 42:
         difficulty_level= LEVEL_8;
         break;
      case 48:
         difficulty_level= LEVEL_9;
         break;
      }
   print_level();
}

void erase_message_window()
{
int  cnt;

   setwritemode(COPY_PUT);
   setcolor(BLUE);
   line(100 - (1 * 8),  100 - (1 * 8), 100 + (20 * 8), 100 - (1 * 8));
   for(cnt= 0; cnt < (10 * 8); cnt++) {
      }
      line(100 - (1 * 8),  100 - (1 * 8) + cnt, 100 + (20 * 8), 100 - (1 * 8) + cnt);
   setcolor(WHITE);

}
```

Figure 8-1 *Continued.*

```
void explosion()
{
int cnt;
   remove_sprites();
   for(cnt= 0; cnt < 25; cnt++) {
      crash();
      delay(1);
      crash1();
      delay(1);
      crash();
      delay(1);
      crash1();
      delay(1);
      crash();
      delay(1);
      crash1();
      delay(1);
      crash();
      delay(1);
      crash1();
      delay(1);
      crash();
      }
}

void remove_sprites()
{
   (M1Ptr->*MEraseImagePtr)();
   (UT1Ptr->*SEraseImagePtr)();
   (UT2Ptr->*SEraseImagePtr)();
   (DT1Ptr->*SEraseImagePtr)();
   (DT2Ptr->*SEraseImagePtr)();
   (LT1Ptr->*SEraseImagePtr)();
   (LT2Ptr->*SEraseImagePtr)();
   (RT1Ptr->*SEraseImagePtr)();
   (RT2Ptr->*SEraseImagePtr)();
   (G_1Ptr->*SEraseImagePtr)();
   (G_2Ptr->*SEraseImagePtr)();
   (G2_1Ptr->*SEraseImagePtr)();
   (G2_2Ptr->*SEraseImagePtr)();
   (G3_1Ptr->*SEraseImagePtr)();
   (G3_2Ptr->*SEraseImagePtr)();
   (G4_1Ptr->*SEraseImagePtr)();
   (G4_2Ptr->*SEraseImagePtr)();
   (G5_1Ptr->*SEraseImagePtr)();
   (G5_2Ptr->*SEraseImagePtr)();

}
```

Part 2

Source code for the bit-plane animation class library

Part 2 is composed of two source-code-laden chapters. This book isn't a programmer's reference, but rather a tutorial on bit-plane sprite animation techniques, so I decided to prototype the sprite management function in Part 1. Placing the function prototypes in the chapters with the sample demonstration programs seemed best. If you want to see how the functions in chapters 9 and 10 are used, refer back to the demonstration programs in chapters 2 through 8.

Chapter 9 contains one assembly module for the sprite management class library. Chapter 10 follows with a presentation of the C++ source code files that contain the class methods used in the bit-plane sprite animation class library.

9

Assembly source code for the sprite library

CHAPTER 9

NOTE that the guts of the bit-plane animation code is contained in the C++ source files in chapter 10. I didn't present the assembly versions of the bit-plane animation code because I couldn't detect any differences between the C++-generated methods and the assembly-generated functions. As C++ (in my opinion) is an easier language to read and maintain than assembly, I chose to write the gut functions in C++.

Assembly sprite management functions

The sprite_a.asm source file contains a method that rotates the bits in a sprite image buffer, and this bit shifting of information in the sprite definition buffer facilitates the smooth horizontal movement of the sprites. See Fig. 9-1 for the source code listing of sprite_a.asm. This file contains shape buffers and a quick routine to expand sprite shapes.

Summary

This chapter presents a single assembly source file that contains bit-plane sprite expansion support. Chapter 10 contains the C++ source modules that are the guts of the bit-plane sprite animation functions.

Figure 9-1 *The source code for SPRITE_A.ASM.*

```
;////////////////////////////////////////
;//
;// sprite_a.asm
;//
;//
;////////////////////////////////////////

   .model   large

   .data

   public _buff6x2
   public _buff48x7
;   public _missile_data
;   public _bullet_data
```

```
    db 128 DUP(?)

_missile_data label byte

                .radix 2
                DB 00000000, 00000000 ; 1
                DB 00111000, 00000000
                DB 01111100, 00000000
                DB 00111000, 00000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 2
                DB 00011100, 00000000
                DB 00111110, 00000000
                DB 00111110, 00000000
                DB 00011100, 00000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 3
                DB 00001110, 00000000
                DB 00011111, 00000000
                DB 00011111, 00000000
                DB 00001110, 00000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 4
                DB 00000111, 00000000
                DB 00001111, 10000000
                DB 00001111, 10000000
                DB 00000111, 00000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 5
                DB 00000011, 10000000
                DB 00000111, 11000000
                DB 00000111, 11000000
                DB 00000011, 10000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 6
                DB 00000001, 11000000
                DB 00000011, 11100000
                DB 00000011, 11100000
                DB 00000001, 11000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 7
                DB 00000000, 11100000
                DB 00000001, 11110000
                DB 00000001, 11110000
                DB 00000000, 11100000
                DB 00000000, 00000000
```

Figure 9-1 *Continued.*

```
                DB 00000000, 00000000 ; 8
                DB 00000000, 01110000
                DB 00000000, 11111000
                DB 00000000, 11111000
                DB 00000000, 01110000
                DB 00000000, 00000000

_bullet_data label byte

                .radix 2
                DB 00000000, 00000000 ; 1
                DB 01100000, 00000000
                DB 01100000, 00000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 2
                DB 00110000, 00000000
                DB 00110000, 00000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 3
                DB 00011000, 00000000
                DB 00011000, 00000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 4
                DB 00001100, 00000000
                DB 00001100, 00000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 5
                DB 00000110, 00000000
                DB 00000110, 00000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 6
                DB 00000011, 00000000
                DB 00000011, 00000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 7
                DB 00000001, 10000000
                DB 00000001, 10000000
                DB 00000000, 00000000

                DB 00000000, 00000000 ; 2
                DB 00000000, 11000000
                DB 00000000, 11000000
                DB 00000000, 00000000
```

```
    .radix 10
    db 128 DUP(?)

_buff6x2        db (6*2) DUP (?)
_buff48x7       db (48*7) DUP (?)

    .CODE

    public @Sprite@shift_sprite$qv

;// shift the sprite image for smooth animation

@Sprite@shift_sprite$qv proc
  push   bp
    mov    bp,sp
    push   si
    mov    si,offset _buff48x7 ; point to offset
    clc                        ; clear carry for
                               ; right shifts
    cld                        ; increment forward
    mov    CX,(48*7)           ; loop 48*7 times
loc1:
    mov    al,[si]             ; get byte from buff48x7
    rcr    al,1                ; shift 1 time
    mov    [si],al             ; stuff back to buffer
    inc    si                  ; get next byte
    loop   loc1                ; loop until done

    pop    si
    mov    sp,bp
    pop    bp
    ret
@Sprite@shift_sprite$qv endp
  end
```

10

C++ source code for the bit-plane animation class library

CHAPTER 10

THIS chapter presents the eight C++ source files that contain the guts of the bit-plane animation classes. As this book has been designed for applications programmers, you don't need to understand the theory and logic of these functions. All you need to know is how to use them. If, however, you want to explore the underlying operations of the class methods, comments in the source code should provide you with a fair amount of information.

The prototypes for the sprite, missile, and bullet methods are contained in the chapter, along with the bit-plane sprite animation demonstration programs from chapters 2 through 8. In addition, the source for the video, playfield, key, and sound classes is also presented in this chapter.

Source code for the sprite class

Figure 10-1 presents SPRITE.CPP, which contains the source code for the methods that manipulate sprite objects.

Source code for the missile class

Figure 10-2 presents MISSILE.CPP, which contains the source code for the methods that manipulate missile objects.

Source code for the bullet class

Figure 10-3 presents BULLET.CPP, which contains the source for the methods that manipulate bullet objects.

Source code for the video class

See Fig. 10-4 for VIDEO.CPP, the source code for the methods that create the video object.

Source code for the playfield class

See Fig. 10-5 for PLAYF.CPP, the source code for the methods that create the playfield object.

Source code for the key class

Figure 10-6 contains KEY.CPP, which contains the source for the methods that create the keyboard object.

Source code for the sound class

Figure 10-7 presents SOUND.CPP, the source for the methods that create the sound object.

Source code for GO.CPP

Figure 10-8 contains the source code for GO.CPP. This file contains the source for global data. (Yes, yes, I know that globals are frowned upon in OOP circles, but I figured that using a few wouldn't hurt.)

Summary

This chapter contains the C++ source code for the following classes: sprite, missile, bullet, playfield, video, key, and sound. These classes form the foundation of the C++ bit-plane animation library. Figure 10-9 presents the library listing for BITPLANE.LIB.

Part 3 of the book is composed of three chapters that focus on the sprite editor. You can use this program to design sprites for use with the bit-plane sprite animation functions presented in Part 2.

Figure 10-1 *The source code for SPRITE.CPP.*

```
////////////////////////////////////////
//
// sprite.cpp
//
// Sprite class member methods
//

#include "goclass.h"

////////////////////////////////////////
// video mode defines if you're using
// your own graphics initialization
// function

#define TEXT80X25MONO     0x02
#define TEXT80X25COLOR    0x03
#define EGA640X350MONO    0x0F
#define EGA640X350COLOR   0x10
#define VGA640X480MONO    0x11
#define VGA640X480COLOR   0x12
#define VGA320X200COLOR   0x13

#define s_off0 0
#define s_off1 48*7*1
#define s_off2 48*7*2
#define s_off3 48*7*3
#define s_off4 48*7*4
#define s_off5 48*7*5
#define s_off6 48*7*6
#define s_off7 48*7*7

extern UCHAR *Gplane1, *Gplane2, *Gplane3, *Gplane4;
extern UCHAR buff48x7[];

// initialize the sprite image

// initialize sprite structure

Sprite::Sprite(UCHAR *dt, UCHAR s_plane)
{
   Sprite::ExpandImage(dt);
   data_table= dt;
   Sprite::plane= s_plane;
   first_move= 0;
  screen_data= (UCHAR *)malloc(48 * 7);
  sprite_data= (UCHAR *)malloc(48 * 7);
};

// destroy the sprite by freeing memory

VOID Sprite::Free()
```

```
{
  free(screen_data);
  free(sprite_data);

};

// relocate the sprite image

VOID Sprite::RelocateImage(int x, int y)
{
   EraseImage();
   MoveImage(x, y);
};

INT Sprite::MissileCollision(missileInfo* mip)
{
int ulx, uly, lrx, lry;          // sprite image
int oulx, ouly, olrx, olry;      // object image

   // if sprite or missile is not visible, return 0

   if((!visible) || (!mip->visible)) {
      return 0;
      }

   // set sprite image upper left

   ulx= current_x;
   uly= current_y;

   // set sprite image lower right

   lrx= ulx + 48;
   lry= uly + 46;

   // set object image upper left

   oulx= mip->x;
   ouly= mip->y;

   // set object image lower right

   olrx= oulx + 8;
   olry= ouly + 8;

   // does upper left object overlap sprite?

   if((oulx >= ulx) &&
      (oulx <= lrx) &&
      (ouly >= uly) &&
      (ouly <= lry)) {
      return 1;
      }
```

Figure 10-1 *Continued.*

```
// does upper right object overlap sprite?

if((olrx >= ulx) &&
   (olrx <= lrx) &&
   (ouly >= uly) &&
   (ouly <= lry)) {
   return 1;
   }

// does lower left object overlap sprite?

if((oulx >= ulx) &&
   (oulx <= lrx) &&
   (olry >= uly) &&
   (olry <= lry)) {
   return 1;
   }

// does lower right object overlap sprite?

if((olrx >= ulx) &&
   (olrx <= lrx) &&
   (olry >= uly) &&
   (olry <= lry)) {
      return 1;
      }

   return(0);
};

INT Sprite::SpriteCollision(spriteInfo* sip)
{
int ulx, uly, lrx, lry;          // sprite image
int oulx, ouly, olrx, olry;      // object image

   // if sprite is not visible, return 0

   if((!visible) || (!sip->visible)) {
      return 0;
      }

   // set sprite image upper left

   ulx= current_x;
   uly= current_y;

   // set sprite image lower right

   lrx= ulx + 48;
   lry= uly + 46;

   // set object image upper left
```

```
oulx= sip->x;
ouly= sip->y;

// set object image lower right

olrx= oulx + 48;
olry= ouly + 46;

// does upper left object overlap sprite?

if((oulx >= ulx) &&
   (oulx <= lrx) &&
   (ouly >= uly) &&
   (ouly <= lry)) {
   return 1;
   }

// does upper right object overlap sprite?

if((olrx >= ulx) &&
   (olrx <= lrx) &&
   (ouly >= uly) &&
   (ouly <= lry)) {
   return 1;
   }

// does lower left object overlap sprite?

if((oulx >= ulx) &&
   (oulx <= lrx) &&
   (olry >= uly) &&
   (olry <= lry)) {
   return 1;
   }

// does lower right object overlap sprite?

if((olrx >= ulx) &&
   (olrx <= lrx) &&
   (olry >= uly) &&
   (olry <= lry)) {
   return 1;
      }

   return 0;
};

SISP Sprite::GetInfo(SISP sip)
{
  sip->x= current_x;
  sip->y= current_y;
  sip->visible= visible;
  return sip;
```

Figure 10-1 *Continued.*

```
};

VOID Sprite::SelectImage(int col)
{

int offset;
UCHAR *srce;

offset= col & 7;
switch(offset) {
   case 0:
      offset=s_off0;
      break;
   case 1:
      offset=s_off1;
      break;
   case 2:
      offset=s_off2;
      break;
   case 3:
      offset=s_off3;
      break;
   case 4:
      offset=s_off4;
      break;
      case 5:
         offset=s_off5;
         break;
      case 6:
         offset=s_off6;
         break;
      case 7:
         offset=s_off7;
         break;
      }

   srce= data_table + offset;
   memcpy(sprite_data, srce, (48 * 7));
};

VOID Sprite::ExpandImage(UCHAR *image_name)
{
int cnt;
   memcpy(buff48x7, image_name, 48*7);
   for(cnt= 1; cnt < 8; ++cnt) {
      shift_sprite();
      memcpy(image_name + (48 * 7 * cnt), buff48x7, 48 * 7);
      }
};

VOID Sprite::MoveImage(int x, int y)
{
   Video::SetPlane(plane);
```

```
   SelectImage(x);
   current_x= x;
   current_y= y;
   visible= 1;
   mvsprite(x, y);
};

// erase the sprite image
VOID Sprite::EraseImage()
{
   Video::SetPlane(plane);
   visible= 0;
   erasprite(current_x, current_y);
};

VOID Sprite::erasprite(int x, int y)
{
UCHAR far *scrn;
int height, i;

   scrn= (UCHAR far *)MK_FP(0xa000, 0);

   scrn+= (80 * y) + (x >> 3);

   for(height= 0; height < 48; height++) {
      for(i= 0; i < 7; i++) {
         *(scrn + i)= 0;
         }
      scrn+= 80;
      }
};

VOID Sprite::mvsprite(int x, int y)
{
UCHAR far *scrn, pel8;
UCHAR far *vid_plane;
int cnt, height, i;
unsigned int offset;

   switch(plane) {
      case 1:
      vid_plane= Gplane1;
      break;
   case 2:
      vid_plane= Gplane2;
      break;
   case 3:
      vid_plane= Gplane3;
      break;
   case 4:
      vid_plane= Gplane4;
      break;
```

Figure 10-1 *Continued.*

```
        }

    scrn= (UCHAR far *)MK_FP(0xa000, 0);

    offset= (80 * y) + (x >> 3);

    scrn+= offset;
    vid_plane+= offset;

       for(cnt= 0, height= 0; height < 48; height++) {
          for(i= 0; i < 7; i++) {
             pel8= *(vid_plane + i);
             pel8|= sprite_data[cnt++];
             *(scrn + i)= pel8;
             }
          scrn+= 80;
          vid_plane+= 80;
          }
};

// save the screen sprite image

VOID Sprite::SaveScreenImage(int x, int y)
{
UCHAR far *scrn;
int cnt, height, i;

   old_x= x;
   old_y= y;

   scrn= (UCHAR far *)MK_FP(0xa000, 0);

   scrn+= (80 * y) + (x >> 3);

   for(cnt= 0, height= 0; height < 48; height++) {
      for(i= 0; i < 7; i++) {
         screen_data[cnt++]= *(scrn + i);
         }
      scrn+= 80;
      }
};

// restore the screen sprite image

VOID Sprite::RestoreScreenImage()
{
UCHAR far *scrn;
int cnt, height, i, x, y;

   x= old_x;
   y= old_y;

   scrn= (UCHAR far *)MK_FP(0xa000, 0);
```

```
    scrn+= (80 * y) + (x >> 3);

    for(cnt= 0, height= 0; height < 48; height++) {
       for(i= 0; i < 7; i++) {
          *(scrn + i)= screen_data[cnt++];
          }
       scrn+= 80;
       }
};
```

The source code for MISSILE.CPP. Figure 10-2

```
////////////////////////////////////////
//
// missile.cpp
//
// Missile class member methods
//
//
//

#include "goclass.h"
extern UCHAR *Gplane1, *Gplane2, *Gplane3, *Gplane4;

// unsigned char MISSILE[6 * 2 * 8];
unsigned char missile_image[12];

UCHAR missile_data[6*2*8]= {
   0, 0,       // 1
   56, 0,
   124, 0,
   124, 0,
   56, 0,
   0, 0,

   0, 0,       // 2
   28, 0,
   62, 0,
   62, 0,
   28, 0,
   0, 0,

0, 0,          // 3
14, 0,
31, 0,
31, 0,
14, 0,
0, 0,

0, 0,          // 4
7, 0,
15, 128,
```

Figure 10-2 *Continued.*

```
    15, 128,
    7, 0,
    0, 0,

    0, 0,       // 5
    3, 128,
    7, 192,
    3, 128,
    0, 0,

    0, 0,       // 6
    1, 192,
    3, 224,
    3, 224,
    1, 192,
    0, 0,

    0, 0,       // 7
    0, 224,
    1, 240,
    1, 240,
    0, 224,
    0, 0,

    0, 0,       // 8
    0, 112,
    0, 248,
    0, 248,
    0, 112,
    0, 0 };

// select missile image data

#define m_off0 0
#define m_off1 6*2*1
#define m_off2 6*2*2
#define m_off3 6*2*3
#define m_off4 6*2*4
#define m_off5 6*2*5
#define m_off6 6*2*6
#define m_off7 6*2*7

// initialize the missile structure

Missile::Missile(UCHAR s_plane)
{
  visible= 0;
  first_move= 0;
  data_table= missile_data;
  plane= s_plane;
  screen_data= (UCHAR *)malloc(128);
  m_data= (UCHAR *)malloc(128);
```

```
  memcpy(m_data, missile_data, 12);
};

MISP Missile::GetInfo(MISP mip)
{
  mip->x= current_x;
  mip->y= current_y;
  mip->visible= visible;
  return mip;
};

// select the missile image

VOID Missile::SelectImage(int col)
{
int offset;
UCHAR *srce;

   offset = col & 7;

   switch(offset)
      {
      case 0:
         offset=m_off0;
         break;
   case 1:
      offset=m_off1;
      break;
   case 2:
      offset=m_off2;
      break;
   case 3:
      offset=m_off3;
      break;
   case 4:
      offset=m_off4;
      break;
   case 5:
      offset=m_off5;
      break;
   case 6:
      offset=m_off6;
         break;
      case 7:
         offset=m_off7;
         break;
      }

   srce= missile_data + offset;
   memcpy(m_data, srce, (6 * 2));
};

// move the missile image
```

Figure 10-2 *Continued.*

```
VOID Missile::MoveImage(int x, int y)
{
   Video::SetPlane(plane);
   SelectImage(x);
   current_x= x;
   current_y= y;
   visible= 1;
   mvmissile(x, y);
};

// erase the missile image

VOID Missile::EraseImage()
{
UCHAR far *v_plane;
UCHAR far *scrn, pel8;
int height, i;
unsigned int offset;
int x, y;

   SetPlane(plane);

   x= current_x;
   y= current_y;

   switch(plane) {
      case 1:
         v_plane= Gplane1;
         break;
      case 2:
         v_plane= Gplane2;
         break;
      case 3:
         v_plane= Gplane3;
         break;
      case 4:
         v_plane= Gplane4;
         break;
      }

   scrn= (UCHAR far *)MK_FP(0xa000, 0);

   offset= (80 * y) + (x >> 3);

   scrn+= offset;
   v_plane+= offset;

   for(height= 0; height < 6; height++) {
      for(i= 0; i < 2; i++) {
         *(scrn + i)= *(v_plane + i);
         }
      scrn+= 80;
      v_plane+= 80;
```

```
      }
   visible= 0;
};

// move the missile image

VOID Missile::mvmissile(int x, int y)
{
UCHAR far *vid_plane;
UCHAR far *scrn, pel8;
int cnt, height, i;
unsigned int offset;

   switch(plane) {
      case 1:
         vid_plane= Gplane1;
         break;
      case 2:
         vid_plane= Gplane2;
         break;
      case 3:
         vid_plane= Gplane3;
         break;
      case 4:
         vid_plane= Gplane4;
         break;
   }

scrn= (UCHAR far *)MK_FP(0xa000, 0);

offset= (80 * y) + (x >> 3);

scrn+= offset;
vid_plane+= offset;

for(cnt= 0, height= 0; height < 6; height++) {
   for(i= 0; i < 2; i++) {
      pel8= *(vid_plane + i);
      pel8|= m_data[cnt++];
      *(scrn + i)= pel8;
      }
   scrn+= 80;
      vid_plane+= 80;
      }
};

VOID Missile::Free()
{
  free(screen_data);
  free(m_data);

};
```

Figure 10-3 *The source code for BULLET.CPP.*

```
////////////////////////////////////////
//
// bullet.cpp
//
// Bullet class member methods
//
//

#include "goclass.h"
extern UCHAR *Gplane1, *Gplane2, *Gplane3, *Gplane4;

unsigned char bullet_image[12];

UCHAR bullet_data[6*2*8]= {
   0, 0,       // 1
   96, 0,
   96, 0,
   0, 0,

   0, 0,       // 2
   48, 0,
   48, 0,
   0, 0,

   0, 0,       // 3
   24, 0,
   24, 0,
   0, 0,

   0, 0,       // 4
   12, 0,
   12, 0,
   0, 0,

   0, 0,       // 5
   6, 0,
   6, 0,
   0, 0,

   0, 0,       // 6
   3, 0,
   3, 0,
   0, 0,

   0, 0,       // 7
   1, 128,
   1, 128,
   0, 0,

   0, 0,       // 8
   0, 192,
```

```
    0, 192,
    0, 0 };

// select bullet image data

#define b_off0 0
#define b_off1 4*2*1
#define b_off2 4*2*2
#define b_off3 4*2*3
#define b_off4 4*2*4
#define b_off5 4*2*5
#define b_off6 4*2*6
#define b_off7 4*2*7

// initialize the bullet structure

Bullet::Bullet(UCHAR s_plane)
{
  visible= 0;
  first_move= 0;
  data_table= bullet_data;
  plane= s_plane;
  screen_data= (UCHAR *)malloc(128);
  b_data= (UCHAR *)malloc(128);
  memcpy(b_data, bullet_data, 8);
};

// select the bullet image

VOID Bullet::SelectImage(int col)
{
int offset;
UCHAR *srce;

   offset = col & 7;

   switch(offset)
      {
      case 0:
         offset=b_off0;
         break;
      case 1:
         offset=b_off1;
         break;
   case 2:
      offset=b_off2;
      break;
   case 3:
      offset=b_off3;
      break;
   case 4:
      offset=b_off4;
      break;
```

Figure 10-3 *Continued.*

```
        case 5:
           offset=b_off5;
           break;
        case 6:
           offset=b_off6;
           break;
        case 7:
           offset=b_off7;
           break;
        }

     srce= bullet_data + offset;
     memcpy(b_data, srce, (6 * 2));
};

// move the bullet image

VOID Bullet::MoveImage(int x, int y)
{
     Video::SetPlane(plane);
     SelectImage(x);
     current_x= x;
     current_y= y;
     visible= 1;
     mvbullet(x, y);
};

// erase the bullet image

VOID Bullet::EraseImage()
{
UCHAR far *v_plane;
UCHAR far *scrn, pel8;
int height, i;
unsigned int offset;
int x, y;

     SetPlane(plane);

     x= current_x;
     y= current_y;

switch(plane) {
     case 1:
        v_plane= Gplane1;
        break;
     case 2:
        v_plane= Gplane2;
        break;
     case 3:
        v_plane= Gplane3;
        break;
     case 4:
```

```
            v_plane= Gplane4;
            break;
         }

   scrn= (UCHAR far *)MK_FP(0xa000, 0);

   offset= (80 * y) + (x >> 3);

   scrn+= offset;
      v_plane+= offset;

      for(height= 0; height < 4; height++) {
         for(i= 0; i < 2; i++) {
            *(scrn + i)= *(v_plane + i);
            }
         scrn+= 80;
         v_plane+= 80;
         }
      visible= 0;
};

// move the bullet image

VOID Bullet::mvbullet(int x, int y)
{
UCHAR far *vid_plane;
UCHAR far *scrn, pel8;
int cnt, height, i;
unsigned int offset;

   switch(plane) {
      case 1:
         vid_plane= Gplane1;
         break;
      case 2:
         vid_plane= Gplane2;
         break;
      case 3:
         vid_plane= Gplane3;
         break;
      case 4:
         vid_plane= Gplane4;
         break;
   }

scrn= (UCHAR far *)MK_FP(0xa000, 0);

offset= (80 * y) + (x >> 3);

scrn+= offset;
vid_plane+= offset;

for(cnt= 0, height= 0; height < 4; height++) {
```

Figure 10-3 *Continued.*

```
for(i= 0; i < 2; i++) {
   pel8= *(vid_plane + i);
   pel8¦= b_data[cnt++];
   *(scrn + i)= pel8;
   }
scrn+= 80;
vid_plane+= 80;
      }
};

VOID Bullet::Free()
{
  free(screen_data);
  free(b_data);
};
```

Figure 10-4 *The source code for VIDEO.CPP.*

```
////////////////////////////////////////
//
// video.cpp
//
//

#include "goclass.h"
extern UCHAR *Gplane1, *Gplane2, *Gplane3, *Gplane4;

// set the ega plane

void Video::SetPlane(unsigned char plane)
{
unsigned ui;

   ui= plane << 8;
   ui¦= 2;
   outpw(0x3c4, ui);
}

// save the video mode

void Video::SaveMode()
{
union REGS ir,or;
   ir.h.ah= 0x0f;
   ir.h.al= 0x10;
   int86(0x10, &ir, &or);
   current_mode= or.h.al;
};

// restore the video mode
```

```
void Video::RestoreMode()
{
union REGS ir,or;
   ir.h.ah= 0;
   ir.h.al= current_mode;
   int86(0x10, &ir, &or);
};

// set the video mode

void Video::SetMode(int v_mode)
{
union REGS ir,or;
   ir.h.ah= 0;
   ir.h.al= (unsigned char)v_mode;
   int86(0x10, &ir, &or);
};
```

The source code for PLAYF.CPP. Figure 10-5

```
////////////////////////////////////////
//
// playf.cpp
//
// PlayField class member methods

#include <stdio.h>
#include <stdlib.h>
#include <mem.h>
#include <alloc.h>

#include "goclass.h"
extern UCHAR *Gplane1, *Gplane2, *Gplane3, *Gplane4;

PlayField::PlayField(int i)
{
unsigned char far *scrn;
unsigned int      offset;
int               cnt;

Number= i;

 Gplane1= plane1= (UCHAR *)malloc(38400);

 if(!plane1) {
    printf("SPRITE_D Module\n");
    printf("Memory Allocation Error -- *PLANE1\n");
    exit(0);
    }
 Gplane2= plane2= (UCHAR *)malloc(38400);
 if(!plane2) {
```

Figure 10-5 *Continued.*

```
        printf("SPRITE_D Module\n");
        printf("Memory Allocation Error -- *PLANE2\n");
        exit(0);
        }
    Gplane3= plane3= (UCHAR *)malloc(38400);
    if(!plane3) {
        printf("SPRITE_D Module\n");
        printf("Memory Allocation Error -- *PLANE3\n");
        exit(0);
        }
    Gplane4= plane4= (UCHAR *)malloc(38400);
    if(!plane4) {
        printf("SPRITE_D Module\n");
        printf("Memory Allocation Error -- *PLANE4\n");
        exit(0);
        }

    offset= 0;
    Video::SetPlane(1);
    scrn= (UCHAR far *)MK_FP(0xa000, 0);
    for(cnt= 0; cnt < 480; cnt++) {
          memcpy((plane1 + offset), (scrn + offset), 80);
          offset+= 80;
        }

    offset= 0;
    Video::SetPlane(2);
    scrn= (UCHAR far *)MK_FP(0xa000, 0);
    for(cnt= 0; cnt < 480; cnt++) {
          memcpy((plane2 + offset), (scrn + offset), 80);
          offset+= 80;
       }

    offset= 0;
    Video::SetPlane(3);
    scrn= (UCHAR far *)MK_FP(0xa000, 0);
    for(cnt= 0; cnt < 480; cnt++) {
          memcpy((plane3 + offset), (scrn + offset), 80);
          offset+= 80;
       }

       offset= 0;
       Video::SetPlane(4);
       scrn= (UCHAR far *)MK_FP(0xa000, 0);
       for(cnt= 0; cnt < 480; cnt++) {
             memcpy((plane4 + offset), (scrn + offset), 80);
             offset+= 80;
          }
};

// clear the playfield

void PlayField::Clear()
```

```
{
char *scrn;
long cnt;

   memset(plane1, 0, (80 * 480));
   memset(plane2, 0, (80 * 480));
   memset(plane3, 0, (80 * 480));
   memset(plane4, 0, (80 * 480));
   scrn= (UCHAR far *)MK_FP(0xa000, 0);
   Video::SetPlane(1);
   memset(scrn, 0, (80 * 480));
   Video::SetPlane(2);
   memset(scrn, 0, (80 * 480));
   Video::SetPlane(3);
   memset(scrn, 0, (80 * 480));
   Video::SetPlane(4);
   memset(scrn, 0, (80 * 480));

};

// destroy the playfield

void PlayField::Destroy()
{
   free(plane1);
   free(plane2);
   free(plane3);
   free(plane4);
}
```

The source code for KEY.CPP.

Figure 10-6

```
////////////////////////////////////////
//
// key.cpp
//
// Keyboard class member methods
//

#include "goclass.h"

Keyboard::Keyboard(int i)
{
  lastKey= i;
};

INT Keyboard::Get()
{
union REGS ir, or;
```

Figure 10-6 *Continued.*

```
  ir.h.ah= 0;
  int86(0x16, &ir, &or);
  lastKey= or.x.ax;
  return(or.x.ax);
};

INT Keyboard::Status()
{
union REGS ir, or;
INT        key;
  _asm {
     mov   ah, 1
     int   16h
     jz    no_key_waiting
     xor   ax, ax
     int   16h
     mov   key, ax
     }
  lastKey= key;
  return key;

no_key_waiting:

  return 0;
};
```

Figure 10-7 *The source code for SOUND.CPP.*

```
//////////////////////////////////////////
//
// sound.cpp
//
// Sound class methods
//
//
//

#include "goclass.h"

//////////////////////////////////////////
//
// Global Variables
//

extern UCHAR *Gplane1, *Gplane2, *Gplane3, *Gplane4;

Sound::Sound(int freq)
{
  tone= freq;
};
```

```
VOID Sound::Restore()
{
  Sound::On(tone);
};

VOID Sound::On(int freq)
{
  tone= freq;
  _asm {
     mov   al, 0b6h
     out   43h, al
     mov   ax, freq
     out   42h, al
     mov   al, ah
     out   42h, al
     in    al, 61h
     or    al, 3
     out   61h, al
     }
};

VOID Sound::Off()
{
  _asm {
     in    al, 61h
     and   al, 0fch
     out   61h, al
     }
};
```

The source code for GO.CPP.

Figure 10-8

```
////////////////////////////////////////
//
// go.cpp
//
// Defines structures used by
// sprite management functions
//

#include <dos.h>
#include <mem.h>

#ifndef UCHAR
  #define UCHAR unsigned char
#endif

#ifndef VOID
  #define VOID void
#endif
```

Figure 10-8 *Continued.*

```
#ifndef INT
  #define INT int
#endif

////////////////////////////////////////
//
// Global Variables
//

UCHAR        *Gplane1, *Gplane2, *Gplane3, *Gplane4;
```

Figure 10-9 *The library listing of the BITPLANE.LIB file.*

```
Publics by module

bullet              size = 982
        Bullet::Bullet(unsigned char)
        Bullet::Free()
        Bullet::SelectImage(int)
        _bullet_data
 Bullet::EraseImage()
 Bullet::MoveImage(int,int)
 Bullet::mvbullet(int,int)
 _bullet_image

go                  size = 16
        _Gplane1
 _Gplane2
        _Gplane3
 _Gplane4

key                 size = 160
        Keyboard::Keyboard(int)
 Keyboard::Get()
        Keyboard::Status()

missile             size = 1047
        Missile::Missile(unsigned char)
 Missile::EraseImage()
        Missile::Free()
 Missile::GetInfo(missileInfo*)
        Missile::MoveImage(int,int)
 Missile::SelectImage(int)
        Missile::mvmissile(int,int)
 _missile_data
        _missile_image

playf               size = 1223
        PlayField::PlayField(int)
```

```
 PlayField::Clear()
        PlayField::Destroy()

sound             size = 159
        Sound::Sound(int)
 Sound::Off()
        Sound::On(int)
 Sound::Restore()

sprite            size = 1641
        Sprite::Sprite(unsigned char*,unsigned char)
 Sprite::EraseImage()
        Sprite::ExpandImage(unsigned char*)
 Sprite::Free()
        Sprite::GetInfo(spriteInfo*)
 Sprite::MissileCollision(missileInfo*)
        Sprite::MoveImage(int,int)
 Sprite::RelocateImage(int,int)
        Sprite::RestoreScreenImage()
 Sprite::SaveScreenImage(int,int)
        Sprite::SelectImage(int)
 Sprite::SpriteCollision(spriteInfo*)
        Sprite::erasprite(int,int)
 Sprite::mvsprite(int,int)

sprite_a          size = 790
        Sprite::shift_sprite()
 _buff48x7
        _buff6x2

video             size = 186
        Video::RestoreMode()
 Video::SaveMode()
        Video::SetMode(int)
 Video::SetPlane(unsigned char)
```

Part 3

The sprite editor

Part 3 presents the instructions for the sprite editor that's used to create the sprite definition source code used in Part 1 of the book. Although the sprite editor is somewhat crude based on today's commercial standards, it nonetheless creates a C source code sprite definition file that can be seamlessly included into source code for bit-plane animation programs. In other words, the sprite editor "gets the job done."

Part 3 is composed of three chapters. Chapter 11 gives directions for using the sprite editor, along with the sprite editor's source code so you can refine the code to your own needs. There are many refinements to the editor you can make. Chapter 12 presents the C source code for the C support functions. There are many useful screen management functions presented in this chapter and you might want to use them in other text-based programs. Chapter 13 presents the assembly source code for additional support functions.

Together, all three parts of this book form a solid foundation for a high-speed bit-plane sprite animation system.

11

The sprite editor

CHAPTER 11

THIS chapter presents the instructions for using the sprite editor, along with the source code for the editor and the make file used to create it. Running the executable program, of course, will produce sprite C source definition files that can be included by other compilers.

As the source code presented in Part 3 has been compiled with Borland C/C++ version 3.1 and 4.0 compilers, so you might need to make minor modifications in the code in order to compile it with other compilers. The process is simple:

1. Change the make file so it invokes your C compiler, assembler, linker, and library manager programs.
2. Build the editor using your new make file.
3. Repair the few errors that occur.

Follow this process and then you'll be ready to tweak the sprite editor's source with your compiler.

How to use the sprite editor

The sprite editor (SPRITE.EXE) is a text-mode-based program that uses the mouse to turn sprite pixels on and off. The editor creates a sprite that's 47 columns wide by 46 rows high, and saves the image in C source code format. You can then include (#include) this C source code in your program during compilation.

You can load previously created sprite images into the sprite editor from the command line. For example, let's say you've created a sprite named CAR. Its sprite file is CAR.C. If you want to edit the CAR sprite, bring it into the editor by using the following command-line message:

```
sprite car.c
```

Clean and simple. Note the following warning; it will save you some possible confusion at some later date. Read and remember it.

WARNING: Each time you save the sprite image back to disk, an additional number is added to the sprite label. Let's say you create a sprite named LEMON and save it back to disk. The buffer containing the sprite definition is named LEMON1. If you save the sprite LEMON again during the same session, it'll be named LEMON11, then LEMON111, etc. So if you ever create a sprite and try to include it into your program and get a missing label error message from your compiler, load the sprite (LEMON.C in this case) into your text editor and make sure the names for the sprite shapes match the names you're using.

While the feature that adds additional numbers to sprite name labels can be bothersome under some circumstances, it also proves somewhat convenient when developing within-sprite animation images.

When you initially run the sprite editor, the left side of the screen has a few messages and buttons that require explanation. I'll take them one by one.

The sprite filename is displayed on the top of the left-hand section of the screen. The name is displayed without the file extension. For example, say I create a two-color sprite using the editor named TANK. The sprite name TANK will appear on the screen. When you save the sprite file back to disk, it will be named TANK.C. The sprites in the file will be named TANK1 and TANK2.

Clicking on the Draw Toggle On box changes the label to Draw Toggle Off. When Draw Toggle On is displayed, pressing the left mouse button in the sprite display field will plot pixels in the currently selected color. When Draw Toggle Off is displayed, pressing the left mouse button will erase the previously plotted sprite pixel.

Clicking on the Save box saves the sprite image source file (.C) back to disk. If you didn't read the previous warning or have forgotten it, read it now. Saving a file a few times affects the sprite's name.

Clicking on the Name box allows you to change the name of the C source file being written to disk. A simple name field entry box

appears on the screen. You enable the name change by pressing Enter.

There's also a box that contains four numbers: 1, 2, 3, and 4. Click on a number with the left mouse button and, when you move into the sprite display field, you'll plot in the color designated by that number.

Returning to earlier discussion, there are four video planes in the 16-color, 640 × 480, VGA video mode. So every time you add a color to a sprite, another data array for the new plane sprite image is added to the source. It's best to create a multicolor sprite image, save the source back to disk, and then have a look at the code with your text editor. You'll see what I mean.

Clicking on the View box rescales the sprite image display so it appears roughly the way it will appear in graphics mode. The only glitch in the View option is that, if the sprite is multicolored, the differentiation of color will be lost. That's because the editor is in the text mode. One big improvement you might consider is to make the View function kick the screen into the graphics mode and display the sprite with the appropriate colors.

To the right of View there are three symbols in boxes: I, +, and –. Clicking on the I symbol inverts the sprite image in the sprite display field. Clicking on the + symbol rotates the sprite 90 degrees clockwise. Clicking on the – symbol rotates the sprite 90 degrees counterclockwise.

These rotation buttons are very useful for creating sprite images that alter according to the direction in which the sprites move on the playfield. These rotation buttons proved invaluable in creating the up, down, left, and right tank image functions in PROG8-1.CPP from chapter 8.

Clicking on the Quit button quits the sprite editor and returns you to DOS. Note that I didn't code a warning into the Quit option to indicate whether or not you saved the sprite image you were creating. You might want to slice into the sprite editor source and add that option.

The sprite display field appears on the right side of the screen, and the entire width of the sprite image appears on the screen. Use the scroll bar on the right to vertically scroll your sprite image. You scroll by placing the rectangular mouse cursor over the up triangle at the top of the scroll bar or the down triangle at the lower end of the scroll bar and pressing the left mouse button. You might want to add action on the scroll bar when you click the mouse on any portion of the vertical scroll bar and not just the top and bottom.

Though a tad kludgy, the sprite editor nonetheless proves very useful in creating multicolor bit-plane sprites. Feel free to add features. I'd love to see any modifications you make to this editor.

Sprite editor C source code

Figure 11-1 presents the source code for SPRITE.C. The functions in this file, in combination with the functions presented in chapters 12 and 13, comprise all you need to build the sprite editor. If you examine the source as you run the program, you'll see how the guts of the sprite editor work. It's really not that complicated.

Figure 11-2 presents the Borland C/C++ version 3.1 make file that's used to build the sprite editor.

Summary

Chapter 11 shows you how to use the sprite editor, and also contains the main source code of the sprite editor. Although the sprite editor is somewhat crude based on today's commercial graphics standards, it nonetheless provides a convenient means of creating multicolored sprites that you can seamlessly integrate into your bit-plane animation programs.

Chapter 12 will present the source code for the C language support functions used by this chapter's sprite editor.

Figure 11-1 *The source code for SPRITE.C.*

```
//////////////////////////////////////////
//
// sprite.c
//
// Description:
//
// Simple Sprite editor
//

// include files here

#include <ctype.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <fcntl.h>
#include <io.h>
#include <sys\stat.h>
#include "tproto.h"

//////////////////////////////////////////
//
// function prototypes
//

void rename_sprite(void);
void invert_sprite(void);
void rotate_sprite_plus90(void);  // top 48 rows clockwise
void rotate_sprite_minus90(void); // top 48 rows counter clockwise
void view_sprite(void);
void write_sprite_to_disk(char *fn);
void write_display_numbers(void);
void display_scroll_bar(void);
void set_file_name(char *fn);
void scroll_up_sprite(void);
void scroll_down_sprite(void);
void display_sprite(void);
int  uc_to_binary(unsigned char *uc);
void set_color_select(void);
int  how_many_sprites(void);
unsigned char *u_to_bin(unsigned char uval, unsigned char val);

//////////////////////////////////////////
//
// defines

#define DISPLAY_FIELD_SIZE (48 * 56)

//////////////////////////////////////////
```

```
//
// global data
//

char     file_name[20];

unsigned char file_loaded= 0;

unsigned int color= 1;

unsigned char sprite[DISPLAY_FIELD_SIZE];
unsigned char sprite_dup[DISPLAY_FIELD_SIZE];

unsigned char colors[4]= { BLUE, GREEN, CYAN, RED };

unsigned char c127= 127;

char          bang= '!';

unsigned char buf196[]= {
   196, 196, 196, 196, 196, 196, 196, 196,
   196, 196, 196, 196, 196, 196, 196, 196,
   196, 196, 196, 196, 196, 196, 196, 196,
   196, 196, 196, 196, 196, 196, 196, 196,
   196, 196, 196, 196, 196, 196, 196, 196,
   196, 196, 196, 196, 196, 196, 196, 196 } ;

unsigned char buf32[]= {
   32, 32, 32, 32, 32, 32, 32, 32,
   32, 32, 32, 32, 32, 32, 32, 32,
   32, 32, 32, 32, 32, 32, 32, 32,
   32, 32, 32, 32, 32, 32, 32, 32,
   32, 32, 32, 32, 32, 32, 32, 32,
   32, 32, 32, 32, 32, 32, 32, 32,
   32, 32, 32, 32, 32, 32, 32, 32,
   32, 32, 32, 32, 32, 32, 32, 32 };

char fill_sprite[] = {
   "0, 0, 0, 0, 0, 0, 0," };

char end_sprite[] = {
   "0, 0, 0, 0, 0, 0, 0 };" };

unsigned char binary[8]= {
   128, 64, 32, 16, 8, 4, 2, 1 };

char slash[40]= {
   47, 47, 47, 47, 47, 47, 47, 47, 47, 47,
   47, 47, 47, 47, 47, 47, 47, 47, 47, 47,
   47, 47, 47, 47, 47, 47, 47, 47, 47, 47,
   47, 47, 47, 47, 47, 47, 47, 47, 47, 47 };
```

Figure 11-1 *Continued.*

```
char space[40]= {
   32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
   32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
   32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
   32, 32, 32, 32, 32, 32, 32, 32, 32, 32 };

unsigned char draw_flag= 1;

char crlf[2]= { 10, 13 };

char ch[]= "unsigned char ";

char br_eq[]= "[48*7*8]= {";

char sprite_name[8];

char data_file_name[15];

char comma_space[2]= { ", "};

char noname[]= "noname.c";

int  display_offset= 0;

int  attr1= 7, attr2= 7, attr3= 7, attr4= 7;
int  cattr1, cattr2, cattr3, cattr4;
int  fcattr1, fcattr2, fcattr3, fcattr4;

////////////////////////////////////////
//
// void rotate_sprite_minus90()
//
// rotate sprite counterclockwise
//
//

void rotate_sprite_minus90()
{
int row, col, ctr1;
unsigned char holder[56];

putch(7);

// move sprite to sprite_dup

for(ctr1= 0; ctr1 < DISPLAY_FIELD_SIZE; ctr1++) {
   sprite_dup[ctr1]= sprite[ctr1];
   }

// move rows to columns

for(row= 0, col= 0; row < 48; row++, col++)  {
```

```
// move top row to holder

for(ctr1= 0; ctr1 < 56; ctr1++) {
   holder[ctr1]= sprite_dup[(row * 56) + ctr1];
   }

// move holder to sprite column
for(ctr1= 47; ctr1 >= 0; ctr1--) {
   sprite[(ctr1 * 56) + col]= holder[ctr1];
      }
   }

   // display the sprite

   display_sprite();

}

////////////////////////////////////////
//
// void rotate_sprite_plus90()
//
// rotate sprite clockwise
//
//

void rotate_sprite_plus90()
{
int row, col, ctr1;
unsigned char holder[56];

   putch(7);

   // move sprite to sprite_dup

   for(ctr1= 0; ctr1 < DISPLAY_FIELD_SIZE; ctr1++) {
      sprite_dup[ctr1]= sprite[ctr1];
      }

   // move rows to columns

   for(row= 0, col= 47; row < 48; row++, col--)  {

   // move top row to holder

   for(ctr1= 0; ctr1 < 56; ctr1++) {
      holder[ctr1]= sprite_dup[(row * 56) + ctr1];
      }

   // move holder to sprite column
   for(ctr1= 0; ctr1 < 48; ctr1++) {
      sprite[(ctr1 * 56) + col]= holder[ctr1];
      }
```

Figure 11-1 *Continued.*

```
        }

    // display the sprite

    display_sprite();

}

////////////////////////////////////////
//
// void invert_sprite()
//
// Turn sprite image upside down
//
//

void invert_sprite()
{
unsigned char holder[56];
int row_top, row_bot, ctr1, ctr2;

    // switch top and bottom rows and move toward center

    for(row_top= 0, row_bot= 47; row_top < 24; row_top++, row_bot--)  {

    // move top row to holder

    for(ctr1= 0; ctr1 < 56; ctr1++) {
       holder[ctr1]= sprite[(row_top * 56) + ctr1];
       }

    // move bottom row to top row

    for(ctr1= 0; ctr1 < 56; ctr1++) {
       sprite[(row_top * 56) + ctr1]= sprite[(row_bot * 56) + ctr1];
       }

    // move holder to bottom row

    for(ctr1= 0; ctr1 < 56; ctr1++) {
       sprite[(row_bot * 56) + ctr1]= holder[ctr1];
          }
       }

    // display the sprite

    display_sprite();

}

////////////////////////////////////////
//
// void rename_sprite()
```

```
//
// Change the sprite name
//

void rename_sprite()
{
char response[30];
int  key;
RECT *R1;
int  attr, offset;

   attr=  mkAttr(WHITE,
                 BLACK,
                 OFF_INTENSITY,
                 OFF_BLINK);

   msoff();
   memset(response, 0, 30);
   strcpy(response, sprite_name);

   R1= setRect(R1, 12, 9, 15, 27);

   saveRect(R1);
   boxRect(R1, D_D_D_D, attr);

vdWrite(12, 11, 15, " Rename Sprite ", attr);

vdWrite(13, 11, 13, " Name:         ", attr1);

key= vdEdit(response, 13, 18, 7, 1, attr1);

if(key == ENTER) {
   // declare sprite name

   memset(sprite_name, 0, 8);

   strcpy(sprite_name, response);
   vdWrite(2, 9, 0, "         ", attr3);
   vdWrite(2, 9, 0, sprite_name, attr3);
   memset(file_name, 0, 20);
   strcpy(file_name, sprite_name);
   strcpy(data_file_name, sprite_name);
   offset= strlen(file_name);
   file_name[offset]= '.';
   data_file_name[offset]= '.';
   offset++;
   file_name[offset]= 'c';
   data_file_name[offset++]= 'd';
   data_file_name[offset++]= 'a';
   data_file_name[offset]= 't';
   }

restRect(R1);
```

Figure 11-1 *Continued.*

```
   dsyRect(R1);

   mson();
}

////////////////////////////////////////
//
// unsigned char *u_to_bin(unsigned char uval, unsigned char val)
//
// Unsigned char to binary conversion. A pointer
// to the "binary" array is returned
//

unsigned char *u_to_bin(unsigned char uval, unsigned char val)
{
unsigned char u8[8], cnt, test;

   memset(u8, 0, 8);

   for(cnt= 0; cnt < 8; cnt++) {
      test= uval &  binary[cnt];
      if(test) {
         u8[cnt]= val;
         }
      }
   return (unsigned char *)u8;
}

////////////////////////////////////////
//
// how_many_sprites()
//
//
// Calculate the number of sprites
// in the buffer

int how_many_sprites()
{
int total, cnt, i1, i2;

   total= 0;

// loop through sprite four times to check for
// the occurrence of each color

for(i1= 1; i1 <= 4; i1++) {
   for(i2= 0; i2 < DISPLAY_FIELD_SIZE; i2++)  {
      if(sprite[i2] == i1) {
         total++;
         i2= DISPLAY_FIELD_SIZE;
         }
      }
   }
```

```
   return total;
}

////////////////////////////////////////
//
// void set_color_select()
//
// Set sprite color select
//

void set_color_select()
{

   msoff();

   vdChar(14, 3, mkToken('1', attr1));
   vdChar(14, 6, mkToken('2', attr1));
   vdChar(14, 9, mkToken('3', attr1));
   vdChar(14, 12, mkToken('4', attr1));

   switch(color) {
      case 1:
         vdChar(14, 3, mkToken('1', cattr1));
         break;
      case 2:
         vdChar(14, 6, mkToken('2', cattr2));
         break;
      case 3:
         vdChar(14, 9, mkToken('3', cattr3));
         break;
      case 4:
         vdChar(14, 12, mkToken('4', cattr4));
         break;
      }
   mson();
}

////////////////////////////////////////
//
// view_sprite()
//
// View the entire sprite
//

void view_sprite()
{
int r, c, offset, vattr;

   // turn mouse off

   msoff();

   // save the screen
```

Figure 11-1 *Continued.*

```
saveScrn();

// clear the screen

for(r= 0; r < 25; r++) {
   for(c= 0; c < 80; c++) {
      vdChar(r, c, mkToken(' ', attr4));
      }
   }

// draw the sprite by row

for(r= 0; r < 24; r++) {

   // draw by column

   for(c= 12; c < 68; c++) {

      // get offset into sprite array

      offset= (r * 112) + (c - 12);

      // check byte and byte below and
      // write proper character
      //
      // top half= 223
      // bot half= 220
      // full    = 219
      //


      // full

      if((sprite[offset]) && (sprite[offset + 56])) {
         vdChar(r, c, mkToken(219, attr4));
         }

      // top

      else if((sprite[offset]) && (!sprite[offset + 56])) {
         vdChar(r, c, mkToken(223, attr4));
         }

      // bottom

      else {
         if((!sprite[offset]) && (sprite[offset + 56])) {
            vdChar(r, c, mkToken(220, attr4));
            }
         }
      }
   }
```

```
// wait here
gtKey();
// restore screen
restScrn();
// turn on the mouse
     mson();
}

////////////////////////////////////////
//
// unsigned char uc_to_binary(unsigned char *uc)
//
// convert 8 BOOLs to 8 bits

int uc_to_binary(unsigned char *uc)
{
int  cnt, u;

   u= 0;
   for(cnt= 0; cnt < 8; cnt++) {
      if(*(uc + cnt)) {
         u|= binary[cnt];
         }
      }

   return u;

}

////////////////////////////////////////
//
// scroll_down_sprite()
//

void scroll_down_sprite()
{
   msoff();
   if(display_offset < 23) {
      display_offset+= 1;
      display_sprite();
      }
   else {
      bleep();
      }
   write_display_numbers();
   mson();
}
```

Figure 11-1 *Continued.*

```
////////////////////////////////////////
//
// scroll_up_sprite()
//

void scroll_up_sprite()
{
   msoff();

   if(display_offset > 0) {
      display_offset-= 1;
      display_sprite();
      }
   else {
      bleep();
      }
   write_display_numbers();
   mson();
}
////////////////////////////////////////
//
// set_file_name(char *fn)
//
// sets the filename buffer
//

void set_file_name(char *fn)
{
int  cnt;

   // declare sprite name

   memset(sprite_name, 0, 8);

   for(cnt= 0; cnt < 8; cnt++) {
      if(*(fn + cnt) == '.') {
         break;
         }
      else {
         sprite_name[cnt]= *(fn + cnt);
         }
      }
}

////////////////////////////////////////
//
// write_sprite_to_disk()
//
// Write sprite source file to disk
//

void write_sprite_to_disk(char *fn)
{
```

```
int handle, num_sprites;
int loop, cnt, cnt1, cflag, nl, number, index;
char buf[25], u2b[8];

   // get the number of sprites

   num_sprites= how_many_sprites();

   // delete old C file

   remove(fn);
   remove(data_file_name);

   handle= open(data_file_name, O_CREAT ¦ O_TRUNC, S_IREAD ¦S_IWRITE);
   write(handle, sprite, DISPLAY_FIELD_SIZE);
   close(handle);

   handle= open(fn, O_CREAT ¦ O_TEXT ¦ O_TRUNC, S_IREAD ¦S_IWRITE);

   // write C comments

   write(handle, slash, 2);
   write(handle, &c127, 1);
   write(handle, slash, 37);
   write(handle, crlf, 1);
   write(handle, slash, 2);
   write(handle, space, 1);
   write(handle, fn, strlen(fn));
   write(handle, crlf, 1);
   write(handle, slash, 2);
   write(handle, crlf, 1);
   write(handle, crlf, 1);

   index= strlen(sprite_name);
   sprite_name[index]= '0';

   for(loop= 0; loop < num_sprites; loop++) {
      write(handle, ch, strlen(ch));

      sprite_name[index]+= 1;

      write(handle, sprite_name, strlen(sprite_name));
      write(handle, br_eq, strlen(br_eq));
      write(handle, crlf, 1);

      nl= 0;
      write(handle, buf32, 3);
      for(cnt= 0; cnt < DISPLAY_FIELD_SIZE; cnt+= 8) {

         // differentiate between different sprites

         cflag= 0;
```

Figure 11-1 *Continued.*

```
for(cnt1= 0; cnt1 < 8; cnt1++) {
   if(sprite[cnt + cnt1] == (loop + 1)) {
      u2b[cnt1]= 1;
      cflag= 1;
      }
   else {
      u2b[cnt1]= 0;
      }
   }

if(cflag) {
   number= uc_to_binary(u2b);
   }
else {
   number= 0;
   }

// write ASCII data to disk

itoa(number, buf, 10);
write(handle, buf, strlen(buf));

// write comma to disk

   write(handle, comma_space, 2);

   // put an ascii NL after seven data elements have been written

   nl++;
   if(nl == 7) {
      nl= 0;
      write(handle, crlf, 1);
      write(handle, buf32, 3);
      }
   }

// write seven batches of 48 rows of sprite data

for(cnt= 0; cnt < 335; cnt++) {
      write(handle, fill_sprite, strlen(fill_sprite));
      write(handle, crlf, 1);
      write(handle, buf32, 3);
      }
   write(handle, end_sprite, strlen(end_sprite));

   write(handle, crlf, 1);
   write(handle, crlf, 1);
   }

// write source file tag

write(handle, slash, 2);
write(handle, &bang, 1);
```

```
write(handle, crlf, 1);
write(handle, slash, 2);
write(handle, space, 1);
write(handle, "End of sprite source", 20);
write(handle, crlf, 1);
write(handle, slash, 40);
write(handle, crlf, 1);

close(handle);

}

////////////////////////////////////////
//
// display_scroll_bar()
//
// Displays the scroll bar on screen
// right

void display_scroll_bar()
{
int  cnt;
   msoff();
   vdChar(0, 74, mkToken(30, attr1));
   vdChar(24, 74, mkToken(31, attr1));
   for(cnt= 1; cnt < 24; cnt++) {
      vdChar(cnt, 74, mkToken(178, attr1));
      }
   mson();
}

////////////////////////////////////////
//
// write_display_numbers()
//
// writes ASCII row numbers in column
//

void write_display_numbers()
{
int  cnt, row;
char buf[25];

   msoff();
   for(row= 1, cnt= display_offset; row < 24; row++, cnt++) {
      itoa((display_offset + row), buf, 10);
      vdWrite(row, 21, 2, buf, attr1);
      vdChar(row, 25, mkToken(179, attr1));
      vdChar(row, 73, mkToken(179, attr1));
      }
   vdWrite(0, 25, 73 - 25, buf196, attr1);
   vdChar(0, 25, mkToken(218, attr1));
```

Figure 11-1 *Continued.*

```
   vdChar(0, 73, mkToken(191, attr1));
   vdWrite(24, 25, 73 - 25, buf196, attr1);
   vdChar(24, 25, mkToken(192, attr1));
   vdChar(24, 73, mkToken(217, attr1));
   mson();
}

////////////////////////////////////////
//
// display_sprite()
//
// display the sprite on the screen
//

void display_sprite()
{
int  r, c, offset, offset1;

   msoff();

   // set offset 1

   offset1= display_offset * 56;

// display sprite data by row

for(r= 1; r < 24; r++) {

   // display sprite data by column

   for(c= 26; c <= 72; c++) {

      // calculate offset into sprite array

      offset= (c - 25) + (r * 56) + offset1;

      // if pixel off

      if(!sprite[offset]) {
         vdChar(r, c, mkToken(' ', attr1));

         }

// else pixel is on

else {
   switch(sprite[offset]) {
      case 1:
         vdChar(r, c, mkToken(219, fcattr1));
         break;
      case 2:
         vdChar(r, c, mkToken(219, fcattr2));
         break;
```

```
            case 3:
               vdChar(r, c, mkToken(219, fcattr3));
               break;
            case 4:
               vdChar(r, c, mkToken(219, fcattr4));
               break;

            }
         }
      mson();
}

////////////////////////////////////////
//
// main()
//
// Program start
//

void main(int argc, char *argv[])
{
char *fn, *fptr, *fptr1, *fptr2;
int  cnt, ret, offset, handle, sprite_count, sprite_processed;
unsigned int num, remaining;
unsigned char uval, *uvalp, val;
int  button, col, row, col1, row1, exit_flag, index, scnt;
char location[25], buf25[25];
RECT *R1, *R2, *R3, *R4, *R5, *R6, *R7, *R8, *R9, *R10, *R11;

   // initialize

   file_loaded= 0;
   sprite_count= 0;
   sprite_processed= 0;

   // make the color attributes

   cattr1= mkAttr(WHITE, BLUE, OFF_INTENSITY, OFF_BLINK);
   cattr2= mkAttr(WHITE, GREEN, OFF_INTENSITY, OFF_BLINK);
   cattr3= mkAttr(WHITE, CYAN, OFF_INTENSITY, OFF_BLINK);
   cattr4= mkAttr(WHITE, RED, OFF_INTENSITY, OFF_BLINK);

   fcattr1= mkAttr(BLUE, WHITE, OFF_INTENSITY, OFF_BLINK);
   fcattr2= mkAttr(GREEN, WHITE, OFF_INTENSITY, OFF_BLINK);
   fcattr3= mkAttr(CYAN, WHITE, OFF_INTENSITY, OFF_BLINK);
   fcattr4= mkAttr(RED, WHITE, OFF_INTENSITY, OFF_BLINK);

   // initialize display buffer to 0

   memset(sprite, 0, DISPLAY_FIELD_SIZE);
```

Figure 11-1 *Continued.*

```
// initialize the attributes

attr1= mkAttr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);
attr2= mkAttr(WHITE, BLUE, OFF_INTENSITY, OFF_BLINK);
attr3= mkAttr(RED, WHITE, OFF_INTENSITY, OFF_BLINK);
attr4= mkAttr(BLUE, WHITE, OFF_INTENSITY, OFF_BLINK);

// initialize video

vidInit();

// clear the screen

scrnClr();

// initialize mouse

ret= msinit();

if(ret == -1) {
   printf("Mouse Not Installed\n");
   printf("Sprite Editor needs the mouse to operate properly.\n");
   printf("Install the mouse and try again.\n");
   printf("Program aborted...\n");
   exit(0);
   }

// determine sprite name

if(argc == 2) {
   set_file_name(argv[1]);
   handle= open(argv[1], O_CREAT ¦ O_TEXT ¦ O_TRUNC, S_IREAD ¦S_IWRITE);
   if(handle == -1) {
      memset(sprite_name, 0, 8);
      strcpy(sprite_name, "noname");
      }

   else {
      memset(data_file_name, 0, 15);
      strcpy(data_file_name, argv[1]);
cnt= 0;
for(;;) {
   if(data_file_name[cnt] != '.') {
      cnt++;
      }
   else {
      cnt++;
      data_file_name[cnt++]= 'd';
      data_file_name[cnt++]= 'a';
      data_file_name[cnt]= 't';
      break;
```

```
        }
    }
handle= open(data_file_name, O_CREAT ¦ O_TRUNC, S_IREAD ¦S_IWRITE);
read(handle, sprite, DISPLAY_FIELD_SIZE);
close(handle);

file_loaded= 1;
}
    }
  else {
     memset(sprite_name, 0, 8);
     strcpy(sprite_name, "noname");
     }

no_display_sprite:

  for(cnt= 0; cnt < strlen(sprite_name); cnt++) {
     sprite_name[cnt]= toupper(sprite_name[cnt]);
     }

  // turn off text cursor

  offCur();

  // turn screen blue

for(cnt= 0; cnt < 25; cnt++) {
   vdAttr(cnt, 0, 80, attr1);
   }

// write vertical bars to screen

for(cnt= 0; cnt < 25; cnt++) {
   vdChar(cnt, 20, mkToken(221, attr1));
   vdChar(cnt, 23, mkToken(222, attr1));
   }
vdWrite(0, 20, 4, buf32, 7);
vdWrite(24, 20, 4, buf32, 7);

// display the scroll bar

display_scroll_bar();

// write display numbers

write_display_numbers();

// display sprite editor name

vdWrite(0, 0, 0, "   Sprite Editor    ", attr2);

// display filename
```

Figure 11-1 *Continued.*

```
vdWrite(2, 3, 0, "Name: ", attr1);
vdWrite(2, 9, 0, sprite_name, attr3);

// display cursor location box

R2= setRect(R2, 4, 2, 7, 18);
boxRect(R2, S_S_S_S, attr1);
vdWrite(4, 3, 15, "Cursor Location", attr1);

// draw on toggle

R3= setRect(R3, 7, 2, 10, 18);
boxRect(R3, S_S_S_S, attr1);
vdWrite(8, 3, 15, "Draw ON Toggle ", attr1);

// save sprite toggle

R4= setRect(R4, 10, 2, 13, 7);
boxRect(R4, S_S_S_S, attr1);
vdWrite(11, 3, 4, "SAVE", attr1);

// rename sprite

R10= setRect(R10, 10, 9, 13, 14);
boxRect(R10, S_S_S_S, attr1);
vdWrite(11, 10, 4, "NAME", attr1);

// view sprite

R5= setRect(R5, 17, 2, 20, 7);
boxRect(R5, S_S_S_S, attr1);
vdWrite(18, 3, 4, "VIEW", attr1);

// invert sprite image

R11= setRect(R11, 17, 9, 20, 15);
boxRect(R11, S_S_S_S, attr1);
vdWrite(18, 10, 1, "I", attr1);
vdChar(17, 11, mkToken(194,attr1));
vdChar(18, 11, mkToken(179,attr1));
vdChar(19, 11, mkToken(193,attr1));
vdWrite(18, 12, 1, "+", attr1);
vdChar(17, 13, mkToken(194,attr1));
vdChar(18, 13, mkToken(179,attr1));
vdChar(19, 13, mkToken(193,attr1));
vdChar(18, 14, mkToken('-',attr1));

// sprite 1 color

R6= setRect(R6, 13, 2, 16, 4);
boxRect(R6, S_S_S_S, cattr1);
vdChar(14, 3, mkToken('1', cattr1));
```

```
R7= setRect(R7, 13, 5, 16, 7);
boxRect(R7, S_S_S_S, cattr2);
vdChar(14, 6, mkToken('2', attr1));

R8= setRect(R8, 13, 8, 16, 10);
boxRect(R8, S_S_S_S, cattr3);
vdChar(14, 9, mkToken('3', attr1));

R9= setRect(R9, 13, 11, 16, 13);
boxRect(R9, S_S_S_S, cattr4);
vdChar(14, 12, mkToken('4', attr1));

 // display quit box

R1= setRect(R1, 22, 2, 25, 7);
boxRect(R1, S_S_S_S, attr1);
vdWrite(23, 3, 4, "QUIT", attr3);

// turn on the mouse

mson();

if(argc == 2) {
   memset(file_name, 0, 20);
   strcpy(file_name, argv[1]);
   }
else {
   strcpy(file_name, noname);
   }

fn= file_name;

// if a file has been loaded

if(file_loaded) {
   display_sprite();
   }

// main loop

exit_flag= aFALSE;
do {
button= msstat(&col, &row);

// adjust mouse location to text coordinates

row>>= 3;
col>>= 3;

if((col1 != col) ¦¦ (row1 != row)) {
   if((row - 1) >= 0   &&
      (row - 1) < 23   &&
      (col - 26) >= 0  &&
```

Figure 11-1 *Continued.*

```
(col - 26)  < 47) {
memset(location, 0, 25);
itoa(col - 26 + 1, location, 10);
vdWrite(5, 4, 0, "X=    ", attr1);
vdWrite(5, 7, 0, location, attr1);
col1= col;

   memset(location, 0, 25);
   itoa(row - 1 + display_offset + 1, location, 10);
   vdWrite(5, 12, 0, "Y=   ", attr1);
   vdWrite(5, 15, 0, location, attr1);
   row1= row;
   }
else {
   vdWrite(5, 12, 0, "Y=    ", attr1);
   vdWrite(5, 4, 0, "X=    ", attr1);
   }
}

// is button press on sprite grid?

if((button == 1) &&
   (row >= 1) && (row <= 23) &&
   ((col >= 26) && (col <= 72))) {
   offset= (col - 25) + (row * 56) + (display_offset * 56);
   if(draw_flag) {
      sprite[offset]= color;
      }
   else {
      sprite[offset]= 0;
      }
   display_sprite();
   }

// toggle draw and erase function

if((button == 1) && (row == 8) && ((col >= 3) && (col <= 17))) {
   if(draw_flag) {
      draw_flag= 0;
      msoff();
      vdWrite(8, 3, 15, "Erase ON Toggle", attr1);
      bleep();
      mson();
      delay(500);
      }
   else {
      draw_flag= 1;
      msoff();
      vdWrite(8, 3, 15, "Draw ON Toggle ", attr1);
      bleep();
      mson();
      delay(500);
      }
```

```
   }

////////////////////////////////////
// button press on up scroll arrow

if((button == 1) && (row == 0) && (col == 74)) {
   scroll_up_sprite();
   }

//////////////////////////////////////
// button press on down scroll arrow

if((button == 1) && (row == 24) && (col == 74)) {
   scroll_down_sprite();
   }

////////////////////////////////////
// set color select

if((button == 1) && (row == 14) && (col == 3)) {
   color= 1;
   set_color_select();
   }

if((button == 1) && (row == 14) && (col == 6)) {
   color= 2;
   set_color_select();
   }

if((button == 1) && (row == 14) && (col == 9)) {
   color= 3;
   set_color_select();
   }

if((button == 1) && (row == 14) && (col == 12)) {
   color= 4;
   set_color_select();
   }

////////////////////////////////////
// view sprite

if((button == 1) && (row == 18) && ((col >= 3) && (col <= 6))) {
   view_sprite();
   }

////////////////////////////////////
// invert sprite

if((button == 1) && (row == 18) && (col == 10)) {
   invert_sprite();
   delay(200);
   }
```

Figure 11-1 *Continued.*

```
////////////////////////////////////
// rotate sprite +90

if((button == 1) && (row == 18) && (col == 12)) {
   rotate_sprite_plus90();
   delay(200);
   }

////////////////////////////////////
// rotate sprite -90

if((button == 1) && (row == 18) && (col == 14)) {
   rotate_sprite_minus90();
   delay(200);
   }

////////////////////////////////////
// save sprite

if((button == 1) && (row == 11) && ((col >= 3) && (col <= 6))) {
   write_sprite_to_disk(fn);
   }

////////////////////////////////////
// name (rename) sprite

if((button == 1) && (row == 11) && ((col >= 10) && (col <= 13))) {
   rename_sprite();
   }

////////////////////////////////////
// quit program

if((button == 1) && (row == 23) && ((col >= 3) && (col <= 6))) {
   exit_flag= aTRUE;
   }

} while(!exit_flag);

// destroy Rect structures

dsyRect(R1);
dsyRect(R2);
dsyRect(R3);
dsyRect(R4);
dsyRect(R5);
dsyRect(R6);
dsyRect(R7);
dsyRect(R8);
dsyRect(R9);
dsyRect(R10);
```

```
   // turn off the mouse cursor

   msoff();

   // clear the screen

   scrnClr();

      // turn on the cursor

      onCur();

}
```

The source code for the make file. Figure 11-2

```
FLAGS= -w3 -ml

OC1= dsyrect.obj vdbox.obj oncur.obj offcur.obj vdedit.obj scloc.obj
OC2= bleep.obj saverect.obj restrect.obj setrect.obj delay.obj
OA1= vidinit.obj mktoken.obj vdchar.obj vdwrite.obj mkattr.obj msoff.obj
OA2= scrnclr.obj mson.obj msinit.obj mvcur.obj  gtkbstat.obj exit_bad.obj
OA3= msstat.obj vdattr.obj vdchr.obj vrdchar.obj s_shape.obj g_shape.obj
OA4= onsound.obj offsound.obj rcloc.obj savescrn.obj restscrn.obj
OA5= sizecur.obj ssizecur.obj rsizecur.obj sizerect.obj gtkey.obj
ALL_O= $(OC1) $(OC2) $(OA1) $(OA2) $(OA3) $(OA4) $(OA5)

sprite.exe: sprite.obj $(ALL_O)
 bcc $(FLAGS) sprite.obj vidtext.lib

sprite.obj: sprite.c
 bcc -c $(FLAGS) sprite.c

dsyrect.obj: dsyrect.c
   bcc -c $(FLAGS) dsyrect.c  >dsyrect.lst
        tlib vidtext -+dsyrect

vdbox.obj: vdbox.c
   bcc -c $(FLAGS) vdbox.c  >vdbox.lst
        tlib vidtext -+vdbox

oncur.obj: oncur.c
   bcc -c $(FLAGS) oncur.c  >oncur.lst
        tlib vidtext -+oncur

offcur.obj: offcur.c
   bcc -c $(FLAGS) offcur.c  >offcur.lst
        tlib vidtext -+offcur

setrect.obj: setrect.c
   bcc -c $(FLAGS) setrect.c  >setrect.lst
        tlib vidtext -+setrect
```

Figure 11-2 *Continued.*

```
bleep.obj: bleep.c
   bcc -c $(FLAGS) bleep.c  >bleep.lst
        tlib vidtext -+bleep

saverect.obj: saverect.c
   bcc -c $(FLAGS) saverect.c  >saverect.lst
        tlib vidtext -+saverect

restrect.obj: restrect.c
   bcc -c $(FLAGS) restrect.c  >restrect.lst
        tlib vidtext -+restrect

vrdchar.obj: vrdchar.c
   bcc -c $(FLAGS) vrdchar.c  >vrdchar.lst
        tlib vidtext -+vrdchar

delay.obj: delay.c
   bcc -c $(FLAGS) delay.c  >delay.lst
        tlib vidtext -+delay

exit_bad.obj: exit_bad.c
   bcc -c $(FLAGS) exit_bad.c  >exit_bad.lst
        tlib vidtext -+exit_bad

sizerect.obj: sizerect.c
   bcc -c $(FLAGS) sizerect.c  >sizerect.lst
        tlib vidtext -+sizerect

vdedit.obj: vdedit.c
   bcc -c $(FLAGS) vdedit.c  >vdedit.lst
        tlib vidtext -+vdedit

scloc.obj: scloc.c
   bcc -c $(FLAGS) scloc.c  >scloc.lst
        tlib vidtext -+scloc

rcloc.obj: rcloc.c
   bcc -c $(FLAGS) rcloc.c  >rcloc.lst
        tlib vidtext -+rcloc

savescrn.obj: savescrn.c
   bcc -c $(FLAGS) savescrn.c  >savescrn.lst
        tlib vidtext -+savescrn

restscrn.obj: restscrn.c
   bcc -c $(FLAGS) restscrn.c  >restscrn.lst
        tlib vidtext -+restscrn

sizecur.obj: sizecur.c
   bcc -c $(FLAGS) sizecur.c  >sizecur.lst
        tlib vidtext -+sizecur
```

```
ssizecur.obj: ssizecur.c
   bcc -c $(FLAGS) ssizecur.c  >ssizecur.lst
        tlib vidtext -+ssizecur

rsizecur.obj: rsizecur.c
   bcc -c $(FLAGS) rsizecur.c  >rsizecur.lst
        tlib vidtext -+rsizecur

vidinit.obj: vidinit.asm
        tasm /mx /Dmdl=3 vidinit.asm
        tlib vidtext -+vidinit

mktoken.obj: mktoken.asm
        tasm /mx /Dmdl=3 mktoken.asm
        tlib vidtext -+mktoken

vdchar.obj: vdchar.asm
        tasm /mx /Dmdl=3 vdchar.asm
        tlib vidtext -+vdchar

vdwrite.obj: vdwrite.asm
        tasm /mx /Dmdl=3 vdwrite.asm
        tlib vidtext -+vdwrite

mkattr.obj: mkattr.asm
        tasm /mx /Dmdl=3 mkattr.asm
        tlib vidtext -+mkattr

msoff.obj: msoff.asm
        tasm /mx /Dmdl=3 msoff.asm
        tlib vidtext -+msoff

scrnclr.obj: scrnclr.asm
        tasm /mx /Dmdl=3 scrnclr.asm
        tlib vidtext -+scrnclr

mson.obj: mson.asm
        tasm /mx /Dmdl=3 mson.asm
        tlib vidtext -+mson

msinit.obj: msinit.asm
        tasm /mx /Dmdl=3 msinit.asm
        tlib vidtext -+msinit

mvcur.obj: mvcur.asm
        tasm /mx /Dmdl=3 mvcur.asm
        tlib vidtext -+mvcur

gtkbstat.obj: gtkbstat.asm
        tasm /mx /Dmdl=3 gtkbstat.asm
        tlib vidtext -+gtkbstat
```

Figure 11-2 *Continued.*

```
msstat.obj: msstat.asm
        tasm /mx /Dmdl=3 msstat.asm
        tlib vidtext -+msstat

vdattr.obj: vdattr.asm
        tasm /mx /Dmdl=3 vdattr.asm
        tlib vidtext -+vdattr

vdchr.obj: vdchr.asm
        tasm /mx /Dmdl=3 vdchr.asm
        tlib vidtext -+vdchr

gtkey.obj: gtkey.asm
        tasm /mx /Dmdl=3 gtkey.asm
        tlib vidtext -+gtkey

g_shape.obj: g_shape.asm
        tasm /mx /Dmdl=3 g_shape.asm
        tlib vidtext -+g_shape

s_shape.obj: s_shape.asm
        tasm /mx /Dmdl=3 s_shape.asm
        tlib vidtext -+s_shape

onsound.obj: onsound.asm
        tasm /mx /Dmdl=3 onsound.asm
        tlib vidtext -+onsound

offsound.obj: offsound.asm
        tasm /mx /Dmdl=3 offsound.asm
        tlib vidtext -+offsound
```

10 12

C language sprite editor support functions

CHAPTER 12

THIS chapter presents the C language source for the support functions that are used by the sprite editor. You might find that many of these functions will prove useful in other programs you're writing. Feel free to use them.

Support function C source code

Figure 12-1 presents the source code for DSYRECT.C.

Figure 12-2 presents the source code for SETRECT.C.

Figure 12-3 presents the source code for SAVERECT.C.

Figure 12-4 presents the source code for RESTRECT.C.

Figure 12-5 presents the source code for SIZERECT.C.

Figure 12-6 presents the source code for BLEEP.C.

Figure 12-7 presents the source code for VDBOX.C.

Figure 12-8 presents the source code for ONCUR.C.

Figure 12-9 presents the source code for OFFCUR.C.

Figure 12-10 presents the source code for VRDCHAR.C.

Figure 12-11 presents the source code for DELAY.C.

Figure 12-12 presents the source code for EXIT_BAD.C.

Figure 12-13 presents the source code for VDEDIT.C.

Figure 12-14 presents the source code for RCLOC.C.

Figure 12-15 presents the source code for SCLOC.C.

Figure 12-16 presents the source code for SAVESCRN.C.

Figure 12-17 presents the source code for RESTSCRN.C.

Figure 12-18 presents the source code for SIZECUR.C.

Figure 12-19 presents the source code for SSIZECUR.C.

Figure 12-20 presents the source code for RSIZECUR.C.

Summary

This chapter contains 20 C source code files. The functions coded in these files are used by the sprite editor. Feel free to use any of them in other programs.

Chapter 13, the final chapter of the book, presents 19 assembly language source files. The functions coded in these files are also used by the sprite editor.

The source code for DSYRECT.C. Figure 12-1

```
//////////////////////////////////////
//
// dsyrect.c
//
// Destroy rectangle structure
//
//////////////////////////////////////

#include "tproto.h"

void dsyRect(RECT *R)
{
   free(R->>image);
   free(R);
}
```

Figure 12-2 *The source code for SETRECT.C.*

```
////////////////////////////////////////
//
// setrect.c
//
// Set rectangular structure
//
//
////////////////////////////////////////

#include <malloc.h>
#include <string.h>
#include "tproto.h"

char b_rect_msg[] = "NULL returned in setRect";

RECT *setRect(RECT *R, int ur, int uc, int lr, int lc)
{
int size;
   R= (RECT *)malloc(sizeof(RECT));
   if(R == 0) {
      exit_bad(b_rect_msg);
      }
   R->ul_row= ur;
   R->ul_col= uc;
   R->lr_row= lr;
   R->lr_col= lc;
   size= sizeRect(R);
   R->image= (unsigned int *)calloc(size, sizeof(int));
   if(R->image == 0) {
      exit_bad(b_rect_msg);
      }
   return(R);
}
```

Figure 12-3 *The source code for SAVERECT.C.*

```
////////////////////////////////////////
//
// saverect.c
//
// Save rectangular text region
//
// WARNING - vidInit MUST be called
//           before this routine!
//
////////////////////////////////////////

#include "tproto.h"

void saveRect(RECT *R)
```

```
{
unsigned int *iptr;
int row;
int column;

   iptr= (unsigned int *)R->image;

   for(row= R->ul_row; row <= R->lr_row; row++) {
      for(column= R->ul_col; column <= R->lr_col; column++) {
         *iptr++= vrdChar(row, column);
         }
      }
}
```

The source code for RESTRECT.C. Figure 12-4

```
////////////////////////////////////////
//
// restrect.c
//
// Restore previously saved rectangle
//
////////////////////////////////////////

#include "tproto.h"

void restRect(RECT *R)
{
unsigned int *iptr;
int row;
int column;

   iptr= (unsigned int *)R->image;

   for(row= R->ul_row; row <= R->lr_row; row++) {
      for(column= R->ul_col; column <= R->lr_col; column++) {
         vdChar(row, column, *iptr++);
         }
      }
}
```

The source code for SIZERECT.C. Figure 12-5

```
////////////////////////////////////////
//
// sizerect.c
//
// Calculate size of memory to hold
// rectangle image
//
//
////////////////////////////////////////
```

Figure 12-5 *Continued.*

```
#include <malloc.h>
#include "tproto.h"

unsigned int sizeRect(RECT *R)
{
int height, width, size;
   height= R->lr_row - R->ul_row;
   width= R->lr_col - R->ul_col;
   ++height;
   ++width;
   size= height * width;
   return(size);
}
```

Figure 12-6 *The source code for BLEEP.C.*

```
//////////////////////////////////////
//
// bleep.c
//
// Makes bleep sound
//
//////////////////////////////////////

#include "tproto.h"

void bleep()
{
int count;

   for(count= 1000; count >? 10; count-= 20) {
        onSound(count);
           Delay(100, 5);
        }

   offSound();
}
```

Figure 12-7 *The source code for VDBOX.C.*

```
//////////////////////////////////////
//
// vdbox.c
//
// Write a box to screen with a
// single-line border
//
//////////////////////////////////////
```

```
#include <stdio.h>
#include "tproto.h"

char vd_blank[80]= { 32, 32, 32, 32, 32, 32, 32, 32,
                     32, 32, 32, 32, 32, 32, 32, 32,
                     32, 32, 32, 32, 32, 32, 32, 32,
                     32, 32, 32, 32, 32, 32, 32, 32,
                     32, 32, 32, 32, 32, 32, 32, 32,
                     32, 32, 32, 32, 32, 32, 32, 32,
                     32, 32, 32, 32, 32, 32, 32, 32,
                     32, 32, 32, 32, 32, 32, 32, 32 };

void boxRect(RECT *R, int box_type, int attr)
{
int row, column;
int top_bot, left_right, ul, ur, ll, lr;

   switch(box_type) {
      case 1:
         top_bot = 196;
         left_right = 186;
         ul = 214;
         ur = 183;
         ll = 211;
         lr = 189;
         break;
   case 2:
         top_bot = 205;
         left_right = 179;
         ul = 213;
         ur = 184;
         ll = 212;
         lr = 190;
         break;
   case 3:
         top_bot = 205;
         left_right = 186;
         ul = 201;
         ur = 187;
         ll = 200;
         lr = 188;
         break;
   default:
         top_bot = 196;
         left_right = 179;
         ul = 218;
         ur = 191;
         ll = 192;
         lr = 217;
         break;
   }
```

Figure 12-7 *Continued.*

```
for(row= R->ul_row; row < R->lr_row - 1; row++) {
   vdWrite(row, R->ul_col, R->lr_col - R->ul_col, vd_blank, attr);
   }

// draw top and bottom

for(column= R->ul_col; column < R->lr_col; ++column) {
   vdChar(R->ul_row, column, mkToken(top_bot, attr));
   vdChar(R->lr_row - 1, column, mkToken(top_bot, attr));
   }

// draw left and right borders

for(row= R->ul_row; row < R->lr_row - 1; ++row) {
   vdChar(row, R->ul_col, mkToken(left_right, attr));
   vdChar(row, R->lr_col, mkToken(left_right, attr));
   }

// plot the four corners

vdChar(R->ul_row, R->ul_col, mkToken(ul, attr));

vdChar(R->ul_row, R->lr_col, mkToken(ur, attr));

   vdChar(R->lr_row - 1, R->ul_col, mkToken(ll, attr));

   vdChar(R->lr_row - 1, R->lr_col, mkToken(lr, attr));
}
```

Figure 12-8 *The source code for ONCUR.C.*

```
//////////////////////////////////////
//
// oncur.c
//
// Turn on (display) text cursor
//
//////////////////////////////////////

#include <stdio.h>

#define C_MASK 0x2000

extern void s_shape(int);
extern int g_shape(void);

void onCur()
{
   s_shape(g_shape() & ~C_MASK);
}
```

The source code for OFFCUR.C. Figure 12-9

```
//////////////////////////////////////
//
// offcur.c
//
// Turn off (remove) text cursor
//
//////////////////////////////////////

#include <stdio.h>
#include "tproto.h"

#define C_MASK 0x2000

extern void s_shape(int);
extern int g_shape(void);

void offCur()
{
   s_shape(g_shape() | C_MASK);
}
```

The source code for VRDCHAR.C. Figure 12-10

```
//////////////////////////////////////
//
// vrdchar.c
//
// Returns a screen token
//
//////////////////////////////////////

#include <stdio.h>
#include "tproto.h"

extern VIDEO *crt;

int vrdChar(int row, int col)
{
long offset;
unsigned int far *scrn;

   scrn= (unsigned int far *)crt->scrn;
   offset= (long)(row * 80) + col;
   return(*(scrn + offset));
}
```

Figure 12-11 *The source code for DELAY.C.*

```
////////////////////////////////////////
//
// delay.c
//
// CPU sensitive delay
//
////////////////////////////////////////

#include "tproto.h"

void Delay(int val1, int val2)
{
int cnt1, cnt2;

   for(cnt1= 0; cnt1 < val1; cnt1++) {
      for(cnt2= 0; cnt2 < val2; cnt2++)  {
         cnt2= cnt2;
         }
      }
}
```

Figure 12-12 *The source code for EXIT_BAD.C.*

```
////////////////////////////////////////
//
// exit_bad.c
//
// Print error message and return to
// DOS
//
////////////////////////////////////////

#include <stdio.h>
#include "tproto.h"

void exit_bad(char *string)
{
   scrnClr();
   mvCur(0, 0);
   printf("NULL POINTER RETURNED - in function\n");
   printf("%s\n\n", string);
   printf("Program ABORT -> Return to DOS\n\n");
   exit(0);
}
```

The source code for VDEDIT.C. Figure 12-13

```
////////////////////////////////////////
//
// vdedit.c
//
// Retrieves string of characters from the
// keyboard
//
////////////////////////////////////////

#include <string.h>
#include "tproto.h"

int vdEdit(char *response,
           int row,
           int column,
           int dlen,
           int opt,
           int attr)
{
int  key;
int  start, stop;
char *rptr;
int  i;
int  ins= 0;
char buf[80];
int  cur, start_column;
int  ret_val;

   // set start column for stopper on left arrow

   start_column= column;

   cur= g_shape();
   sCloc();
   onCur();
   *(response + dlen)= 0;
   switch(opt) {
      case LOWER:
          strlwr(response);
          break;
      case NAME:
          strlwr(response);
          *response= toupper(*response);
          break;
      case UPPER:
          strupr(response);
          break;
      }

for(i= 0;i < dlen; i++) {
   buf[i]= response[i];
   }
```

Figure 12-13 *Continued.*

```
start= column;
stop= start + dlen;
vdAttr(row, column, dlen, attr);
if(*response) {
   vdWrite(row, column, 0, response, attr);
   }
mvCur(row, column+= strlen(response));
rptr= response + strlen(response);

key= gtKey();

switch(key) {
   case ESCAPE:
   case ENTER:
   case F1:
   case F2:
   case F3:
   case F4:
   case F5:
   case F6:
   case F7:
   case F8:
   case F9:
   case F10:
case UP_ARROW:
case DOWN_ARROW:
case TAB:
   s_shape(cur);
   rCloc();
   return(key);
case HOME:
case PGUP:
   memset(response, 0, dlen + 1);
   rptr= response;
   vdWrite(row, start, dlen, response, attr);
   column= start;
   mvCur(row,column);
   break;
case CNTL_LEFTA:
   while (*--rptr!=' ');
   rptr++;
   if(start_column<start+(int)(rptr-response)) {
      mvCur(row,column=start+(int)(rptr-response));
      }
   else {
      column= start_column;
      mvCur(row, start_column);
      }
   vdWrite(row, start, dlen, response, attr);
   break;
case LEFT_ARROW:
   if(start_column < column) {
```

```
            mvCur(row, --column);
            rptr--;
            }
      case END:
      case INSERT:
      case DELETE:
      case RIGHT_ARROW:
      case PGDN:
         vdWrite(row, start, dlen, response, attr);
         break;
      default:
         switch(opt) {
            case LOWER:
               key= tolower(key);
               break;
            case UPPER:
               key= toupper(key);
               break;
            }
         key&= 0x00ff;
         if((key>=0x20)&&(key<=0x7e)) {
            memset(response, 0, dlen + 1);
            rptr= response;
            *rptr++= (char)key;
            vdWrite(row, start, dlen, response, attr);
            column= start + 1;
            mvCur(row, column);
            }
         if(key == aBS) {
            if(column > start) {
               rptr--;
               *rptr= 0;
               vdWrite(row, start, dlen, response, attr);
               mvCur(row, --column);
               }
            }
      }

   do {
      if(opt == NAME) {
         *response = toupper(*response);
         vdWrite(row,start,dlen,response,attr);
         }
      key = gtKey();
      switch(key) {
         case F1:
         case F2:
         case F3:
         case F4:
         case F5:
         case F6:
         case F7:
         case F8:
         case F9:
```

Figure 12-13 *Continued.*

```
case F10:
case UP_ARROW:
case DOWN_ARROW:
case TAB:
case ENTER:
   s_shape(cur);
   rCloc();
   return key;
case ESCAPE:
   s_shape(cur);
   rCloc();
   for (i=0;i<dlen;i++)
       response[i]=buf[i];
   return key;
case CNTL_G:
case DELETE:
   for (i=0;i<stop-column+1;i++)
       *(rptr+i)=*(rptr+1+i);
   vdWrite(row,start,dlen,response,attr);
   break;
case CNTL_T:
   while (*rptr&&*rptr!=' ')
       for (i=0;i<stop-column+1;i++)
           *(rptr+i)=*(rptr+1+i);
   while (*rptr&&*rptr==' ')
       for (i=0;i<stop-column+1;i++)
           *(rptr+i)=*(rptr+1+i);
   vdWrite(row,start,dlen,response,attr);
   break;
case CNTL_END:
   memset(rptr,0,stop-column);
   vdWrite(row,start,dlen,response,attr);
   break;
case LEFT_ARROW:
   if(start_column<column) {
      mvCur(row,--column);
      rptr--;
      }
   break;
case CNTL_LEFTA:
   if(rptr == response) {
      break;
      }
   while(*--rptr == ' ' && (int)(rptr - response) > 0);
   while(*--rptr != ' ' && (int)(rptr - response) > 0);
   if((int)(rptr - response) > 0) {
      rptr++;
      }
   mvCur(row, column= start + (int)(rptr - response));
   break;
case CNTL_RIGHTA:
   if(*rptr) {
```

```
            while(*++rptr != ' ' && *rptr);
            }
         if(*rptr) {
            while(*++rptr == ' ' && *rptr);
            }
         mvCur(row,column= start + (int)(rptr - response));
         break;
      case RIGHT_ARROW:
         if(*rptr) {
            mvCur(row,++column);
            rptr++;
            }
         break;
      case CNTL_BS:
         memset(response, 0, dlen + 1);
         rptr= response;
         vdWrite(row, start, dlen, response, attr);
         column= start;
         mvCur(row, column);
         break;
      case HOME:
         mvCur(row, column= start);
         rptr= response;
         break;
      case END:
         mvCur(row, column= start + strlen(response));
         rptr= response + strlen(response);
         break;
      case CNTL_H:
      case BS:
         if(column>start) {
            rptr--;
            for(i= 0;i < stop - column + 1; i++) {
               *(rptr + i)= *(rptr + 1 + i);
               }
            vdWrite(row, start, dlen, response, attr);
            mvCur(row, --column);
            }
         else {
            bleep();
            }
         break;
      case INSERT:
         if(ins) {
            ins= 0;
            rsizeCur();
            }
         else {
            ins= 1;
            ssizeCur();
            sizeCur(0, 7);
            }
      default:
         switch (opt) {
```

Figure 12-13 *Continued.*

```
case NAME:
case LOWER:
   key= tolower(key);
   break;
case UPPER:
   key= toupper(key);
   break;
            }
key &=0x00ff;
if((key>=0x20)&&(key<=0x7d)) {
   if(ins) {
      if(strlen(response) < dlen) {
         for(i= dlen; i > (rptr - response); i--) {
            response[i] = response[i-1];
            }
         vdWrite(row,start,dlen,response,attr);
         }
      else {
         bleep();
         break;
         }
      }
   if(column < stop) {
      vdChar(row, column, (attr<< 8) + key);
      *rptr++= (char)key;
      column++;
      mvCur(row,column);
      }
   else {
      bleep();
      }
   }
}
} while(1);
}
```

Figure 12-14 *The source code for RCLOC.C.*

```
////////////////////////////////////////
//
// rcloc.c
//
// Restore previously saved cursor location
//
////////////////////////////////////////

#include <dos.h>
#include "tproto.h"

extern CUR_LOCATION *c_loc;

void rCloc()
{
   mvCur(c_loc->row, c_loc->column);
}
```

The source code for SCLOC.C.

Figure 12-15

```
////////////////////////////////////////
//
// scloc.c
//
// Save cursor location to memory
//
////////////////////////////////////////

#include <dos.h>
#include "tproto.h"

CUR_LOCATION *c_loc;

void sCloc()
{
union REGS ir,or;

   c_loc= (CUR_LOCATION *)calloc(sizeof(CUR_LOCATION), sizeof(int));
   ir.h.ah= 0x03;
   ir.h.bh= 0x00;
   int86(0x10, &ir, &or);
   c_loc->row= or.h.dh;
   c_loc->column= or.h.dl;
}
```

The source code for SAVESCRN.C.

Figure 12-16

```
////////////////////////////////////////
//
// savescrn.c
//
// Save screen to memory
//
// WARNING - vidInit MUST be called
//           before this routine!
//
////////////////////////////////////////

#include "tproto.h"

extern unsigned int SCRN_MEM[80*25];

void saveScrn()
{
unsigned int *iptr;
int row;
int column;

   iptr= SCRN_MEM;

   for(row= 0; row < 25; row++) {
```

Figure 12-16 *Continued.*

```
        for(column= 0; column < 80; column++)  {
           *iptr++= vrdChar(row, column);
           }
        }
}
```

Figure 12-17 *The source code for RESTSCRN.C.*

```
////////////////////////////////////////
//
// restscrn.c
//
// Restore previously saved screen
//
////////////////////////////////////////

#include "tproto.h"

unsigned int SCRN_MEM[80*25];

void restScrn()
{
unsigned int *iptr;
int row;
int column;

   iptr= SCRN_MEM;

   for(row= 0; row < 25; row++) {
      for(column= 0; column < 80; column++) {
         vdChar(row, column, *iptr++);
         }
      }
}
```

Figure 12-18 *The source code for SIZECUR.C.*

```
////////////////////////////////////////
//
// sizecur.c
//
// Set cursor size
//
//
////////////////////////////////////////

#include <dos.h>
#include "tproto.h"

void sizeCur(int start, int end)
{
```

```
union REGS ir,or;
int token;
   token= mkToken(end, start);
   ir.h.ah= 0x01;
   or.x.cx= token;
   int86(0x10, &ir, &or);
}
```

The source code for SSIZECUR.C. Figure 12-19

```
////////////////////////////////////////
//
// ssizecur.c
//
// Save default cursor shape
//
////////////////////////////////////////

#include <dos.h>
#include "tproto.h"

extern int CUR_SHAPE;

void ssizeCur()
{
union REGS ir,or;

   ir.h.ah= 0x03;
   ir.h.bh= 0x00;
   int86(0x10, &ir, &or);
   CUR_SHAPE=  or.x.cx;
}
```

The source code for RSIZECUR.C. Figure 12-20

```
////////////////////////////////////////
//
// rsizecur.c
//
// Restore previously saved cursor size
//
////////////////////////////////////////

#include <dos.h>
#include "tproto.h"

int CUR_SHAPE;

void rsizeCur()
{
```

Figure 12-20 *Continued.*

```
union REGS ir, or;

   ir.h.ah= 0x01;
   ir.x.cx= CUR_SHAPE;
   int86(0x10, &ir, &or);
}
```

10

13

Assembly language sprite editor support functions

CHAPTER 13

CHAPTER 13, the final chapter in the book, presents 19 assembly language source files. These files contain functions that are used by the sprite editor. Feel free to use any of them in any programs you're writing.

Support function assembly language support code

Figure 13-1 presents the source code for VIDINIT.ASM.

Figure 13-2 presents the source code for GTKBSTAT.ASM.

Figure 13-3 presents the source code for GTKEY.ASM.

Figure 13-4 presents the source code for G_SHAPE.ASM.

Figure 13-5 presents the source code for MKATTR.ASM.

Figure 13-6 presents the source code for MKTOKEN.ASM.

Figure 13-7 presents the source code for MSOFF.ASM.

Figure 13-8 presents the source code for MSINIT.ASM.

Figure 13-9 presents the source code for MSON.ASM.

Figure 13-10 presents the source code for MSSTAT.ASM.

Figure 13-11 presents the source code for MVCUR.ASM.

Figure 13-12 presents the source code for OFFSOUND.ASM.

Figure 13-13 presents the source code for ONSOUND.ASM.

Figure 13-14 presents the source code for S_SHAPE.ASM.

Figure 13-15 presents the source code for SCRNCLR.ASM.

Figure 13-16 presents the source code for VDATTR.ASM.

Figure 13-17 presents the source code for VDCHAR.ASM.

Figure 13-18 presents the source code for VDCHR.ASM.

Figure 13-19 presents the source code for VDWRITE.ASM.

Summary

Chapter 13 contains 19 assembly source code modules. The functions in these assembly modules are used by the sprite editor.

The source code for VIDINIT.ASM. Figure 13-1

```
;/////////////////////////////////////
;//
;// vidinit.asm
;//
;// vidInit();
;//
;/////////////////////////////////////

        dosseg
IF   mdl eq 1
        .model small,c
ELSEIF  mdl eq 2
        .model medium,c
ELSE
        .model large,c
ENDIF
         v struc                    ; v STRUCT MUST MATCH
          mode  dw      0           ; data struct of VIDEO
          wid   dw      0           ; struct in tstruch.h
          pag   dw      0
          scrn  dw      0,0
         v ends

        public        SCRNSEG,crt,VID_PORT,SPARKLE_FLAG

        .data
SPARKLE_FLAG    dw      0       ; No sparkle fix default
VID_PORT        dw      0       ; video controller status port
SCRNSEG         dw      0       ; int holds scrn seg
crt             dw      0,0     ; pointer to VIDEO struct
vid             v       <>      ; structure declaration
```

Figure 13-1 *Continued.*

```
        public      vidInit
        .code
vidInit proc

; move offset of pointer to video structure to global
        mov     crt+2,_DATA
        mov     crt,offset vid  ; addr of struct -> _crt

; get video mode -> int 10h func 15

        xor     cx,cx           ; cx -> 0
        mov     ah,15           ; BIOS get mode
        int     10h             ; BIOS int
        mov     cl,al           ; mode -> vid.mode
        mov     [vid.mode],cx
        mov     cl,ah           ; row wid -> vid.width
        mov     [vid.wid],cx
        mov     cl,bh           ; page -> vid.pag
        mov     [vid.pag],cx

; prep structure for mono or color

        cmp     al,7            ; is mono?
        je      ismono          ; yes ->jump
        mov     ax,cx
        mov     cl,8
        shl     ax,cl
        add     ax,0b800h
        mov     VID_PORT,03dah  ; stat color controller port
        mov     SCRNSEG,ax      ; color scrn seg
        mov     [vid.scrn],00h  ; far * offset
        mov     [vid.scrn+2],ax ; far * seg
        jmp     videxit

ismono:
        mov     VID_PORT,03bah  ; stat mono controller port
        mov     SCRNSEG,0b000h  ; mono scrn seg
        mov     [vid.scrn],00h  ; far * offset
        mov     [vid.scrn+2],0b000h ; far * seg

videxit:
        ret

vidInit endp
        end

;////////////////////////////////////////
```

The source code for GTKBSTAT.ASM. Figure 13-2

```
;//////////////////////////////////////
;//
;// gtkbstat.asm
;//
;// Checks the status of keyboard and
;// returns the 16-bit char and scan code
;// on key press and a 0 on no key waiting
;//
;// key= gtKBstat();
;//
;//////////////////////////////////////

        dosseg
IF   mdl eq 1
        .model small,c
ELSEIF  mdl eq 2
        .model medium,c
ELSE
        .model large,c
ENDIF
        .code

        public  gtKBstat
gtKBstat proc
        mov         AH,1        ; kb stat function
        int         16H         ; keybd int
        jnz         yeskey      ; jmp on no key waiting
        mov         AX,0        ; no key wait return 0
        jmp         keyexit
yeskey:                         ; otherwise return scan & char
        mov         AH,0
        int         16H
keyexit:
        ret
gtKBstat endp
        end

;//////////////////////////////////////
```

The source code for GTKEY.ASM. Figure 13-3

```
;//////////////////////////////////////
;//
;// gtkey.asm
;//
;// Waits for a key press and returns
;// the key code for the key press
;//
;// key= gtKey();
;//
;//////////////////////////////////////
```

Figure 13-3 *Continued.*

```
        dosseg
IF   mdl eq 1
        .model small,c
ELSEIF  mdl eq 2
        .model medium,c
ELSE
        .model large,c
ENDIF
        .code

        .code

        public  gtKey

gtKey   proc
        mov     AH,0
        int     16H
        ret

gtKey   endp

        end
;//////////////////////////////////////////
```

Figure 13-4 *The source code for G_SHAPE.ASM.*

```
;//////////////////////////////////////////
;//
;// g_shape.asm
;//
;// Gets cursor shape
;//
;// shape= g_shape();
;//
;//
;//////////////////////////////////////////

        dosseg
IF   mdl eq 1
        .model small,c
ELSEIF  mdl eq 2
        .model medium,c
ELSE
        .model large,c
ENDIF
        .code

        public  g_shape

g_shape proc
        mov     ah,3    ; GET_CURS
```

```
        int     10h     ; video bios
        mov     ax,cx   ; shape -> ax
        ret
g_shape endp
        end

;////////////////////////////////////////
```

The source code for MKATTR.ASM. Figure 13-5

```
;////////////////////////////////////////
;//
;// mkattr.asm
;//
;// Sets a character display attribute
;//
;// attr= mkAttr(int fore, int back, int intensity, int blink);
;//
;////////////////////////////////////////

        dosseg
IF   mdl eq 1
        .model small,c
ELSEIF  mdl eq 2
        .model medium,c
ELSE
        .model large,c
ENDIF
        .code

        public mkAttr
        .code
mkAttr  proc
        arg     fore_c:word,back_c:word,inten_t:word,blink_t:word
        xor     AX,AX       ; 0 -> AX
        mov     ax,back_c   ; back ground color to AL
        mov     CL,4        ; prep shift 4 left
        shl     ax,CL       ; means AL * 16
        or      ax,fore_c   ; or foreground color
        or      ax,inten_t  ; or intensity
        or      ax,blink_t  ; or blink
        ret

mkAttr  endp
        end

;////////////////////////////////////////
```

Figure 13-6 *The source code for MKTOKEN.ASM*

```
;/////////////////////////////////////////
;//
;// mktoken.asm
;//
;// Converts screen char and attribute into 16-bit token
;//
;// token= mkToken(char ch, int attr);
;//
;// token == the 16-bit screen token
;//
;/////////////////////////////////////////

        dosseg
IF   mdl eq 1
        .model small,c
ELSEIF  mdl eq 2
        .model medium,c
ELSE
        .model large,c
ENDIF
        .code
        public  mkToken

mkToken proc
        arg     char:byte,attr:byte
        mov     al,char
        mov     ah,attr
        ret

mkToken endp
        end

;/////////////////////////////////////////
```

Figure 13-7 *The source code for MSOFF.ASM.*

```
;/////////////////////////////////////////
;//
;// msoff.asm
;//
;// Removes the mouse from the screen
;//
;// msoff();
;//
;/////////////////////////////////////////

        dosseg
IF   mdl eq 1
        .model small,c
ELSEIF  mdl eq 2
```

```
        .model medium,c
ELSE
        .model large,c
ENDIF
        .code
        public  msoff

msoff   proc
        mov     ax,2
        int     33h
        ret
msoff   endp
        end

;/////////////////////////////////////////
```

The source code for MSINIT.ASM. Figure 13-8

```
;/////////////////////////////////////////
;//
;// msinit.asm
;//
;// Initializes the mouse
;//
;// buttons= msinit();
;//
;// if buttons == ffffh, then mouse not installed
;// else buttons == number of buttons on mouse
;//
;/////////////////////////////////////////

        dosseg
IF   mdl eq 1
        .model small,c
ELSEIF  mdl eq 2
        .model medium,c
ELSE
        .model large,c
ENDIF
        .code
        public  msinit

msinit  proc
        xor     ax,ax
        int     33h
        cmp     ax,0
        je      nomouse
        mov     ax,bx
        ret
nomouse:
        mov     AX,0ffffh
        ret
```

Figure 13-8 *Continued.*

```
msinit  endp
        end

;//////////////////////////////////////
```

Figure 13-9 *The source code for MSON.ASM.*

```
;//////////////////////////////////////////
;//
;// mson.asm
;//
;// Displays the mouse on the screen
;//
;// mson();
;//
;//////////////////////////////////////////

        dosseg
IF   mdl eq 1
        .model small,c
ELSEIF  mdl eq 2
        .model medium,c
ELSE
        .model large,c
ENDIF
        .code
        public  mson

mson    proc
        mov     ax,1
        int     33h
        ret
mson    endp
        end

;//////////////////////////////////////
```

Figure 13-10 *The source code for MSSTAT.ASM.*

```
;//////////////////////////////////////////
;//
;// msstat.asm
;//
;// Returns the status of the mouse
;//
;// button= msstat(int x, int y);
;//
;// if button == 1 left mouse button press
;// if button == 2 right mouse button press
;// x == mouse x position in pixel coordinates
;// y == mouse y position in pixel coordinates
```

```
;//
;//////////////////////////////////////

        dosseg
IF   mdl eq 1
        .model small,c
ELSEIF  mdl eq 2
        .model medium,c
ELSE
        .model large,c
ENDIF
        .code
        public  msstat

msstat  proc
        arg     x:ptr,y:ptr
        mov     ax,3
        int     33h
        mov     ax,bx
IF mdl eq 3
        push    ds
        lds     bx,x
        mov     ds:[bx],cx
        lds     bx,y
        mov     ds:[bx],dx
        pop     ds
ELSE
        mov     bx,x
        mov     word ptr [bx],cx
        mov     bx,y
        mov     word ptr [bx],dx
ENDIF
        ret

msstat endp
        end

;//////////////////////////////////////
```

The source code for MVCUR.ASM.

Figure 13-11

```
;//////////////////////////////////////
;//
;// mvcur.asm
;//
;// Moves the text cursor
;//
;// mvCur(int row, int col);
;//
;//////////////////////////////////////

        dosseg
```

Figure 13-11 *Continued.*

```
IF   mdl eq 1
        .model small,c
ELSEIF  mdl eq 2
        .model medium,c
ELSE
        .model large,c
ENDIF
        .code

        public  mvCur
mvCur proc
        arg     row:byte,column:byte
        mov     ah,0fh
        int     10h
        xor     al,al
        mov     dh,row
        mov     dl,column
        mov     ah,2            ; vid mv cur
        int     10h             ; BIOS video int
        ret
mvCur   endp
        end

;////////////////////////////////////////
```

Figure 13-12 *The source code for OFFSOUND.ASM.*

```
;////////////////////////////////////////
;//
;// offsound.asm
;//
;// Silences the PC speaker
;//
;// offSound();
;//
;////////////////////////////////////////

        dosseg
IF   mdl eq 1
        .model small,c
ELSEIF  mdl eq 2
        .model medium,c
ELSE
        .model large,c
ENDIF
        .code

        public  offSound
offSound proc
        in          al,61h          ; disable speaker output via timer
        and         al,0fch
```

```
        out         61h,al
        ret
offSound endp
        end

;////////////////////////////////////////
```

The source code for ONSOUND.ASM. Figure 13-13

```
;////////////////////////////////////////
;//
;// onsound.asm
;//
;// Starts a tone from the PC speaker
;//
;// onSound();
;//
;////////////////////////////////////////

        dosseg
IF   mdl eq 1
        .model small,c
ELSEIF  mdl eq 2
        .model medium,c
ELSE
        .model large,c
ENDIF
        .code

         public  onSound
onSound proc
        arg     tone:word
        mov     al,0b6h         ; tell timer prep for new sound
        out     43h,al
        mov     AX,tone         ; new tone to timer, LSB
        out     42h,al
        mov     al,Ah           ; MSB -> LSB
        out     42h,al          ; LSB -> timer
        in      al,61h          ; enable speaker output via timer
        or      al,3
        out     61h,al
        ret
onSound endp
        end

;////////////////////////////////////////
```

Figure 13-14 *The source code for S_SHAPE.ASM.*

```
;/////////////////////////////////////////
;//
;// s_shape.asm
;//
;// Sets cursor shape
;//
;// s_shape(int shape);
;//
;//
;/////////////////////////////////////////

        dosseg
IF   mdl eq 1
        .model small,c
ELSEIF  mdl eq 2
        .model medium,c
ELSE
        .model large,c
ENDIF
        .code

        public  s_shape
s_shape proc
        arg     shape:word
        mov     cx,shape        ; cur shape -> cx
        mov     ah,1            ; set chape func
        int     10h             ; video bios
        ret
s_shape endp
        end

;/////////////////////////////////////////
```

Figure 13-15 *The source code for SCRNCLR.ASM.*

```
;/////////////////////////////////////////
;//
;// scrnclr.asm
;//
;// Clears the screen
;//
;// scrnClr();
;//
;/////////////////////////////////////////

        dosseg
IF   mdl eq 1
        .model small,c
ELSEIF  mdl eq 2
        .model medium,c
```

```
ELSE
        .model large,c
ENDIF
        .code

        public  scrnClr
scrnClr proc
        xor         ax,ax           ; lines to scroll 0
        mov         cx,ax           ; UL row to 0
                                    ; UL column to 0
        mov         dh,24           ; LR row to 24
        mov         dl,79           ; LR column to 70
        mov         bh,7            ; fore->white, back->black
        mov         ah,6            ; vid scroll up function
        int         10h             ; bios do it
        mov         ah,15           ; state int -> page to bh
        int         10h
        mov         dx,0            ; row & col to 0
        mov         ah,2            ; reset cursor position
        int         10h
        ret
scrnClr endp
        end
```

The source code for VDATTR.ASM. Figure 13-16

```
;//////////////////////////////////////
;//
;// vdattr.asm
;//
;// Changes display attribute without
;// altering the character
;//
;// vdAttr(int row, int col, int length, int attr);
;//
;//////////////////////////////////////

        dosseg
IF   mdl eq 1
        .model small,c
ELSEIF  mdl eq 2
        .model medium,c
ELSE
        .model large,c
ENDIF

        extrn   SCRNSEG:word

        .code
        public   vdAttr
vdAttr  proc
        arg     prow:byte,pcol:byte,plen:word,pattr:byte
```

Figure 13-16 *Continued.*

```
        push    di
        push    si
        mov     cx,word ptr dgroup:SCRNSEG
        mov     es,cx           ; reset extra seg
        xor     ax,ax           ; 0 -> ax
        mov     al,prow         ; row -> al
        mov     bl,160          ; 80 chars wide * 2
        mul     bl              ; row * scrn width  -> ax
        mov     cl,pcol         ; column to cl
        xor     ch,ch           ; 0 -> ch
        shl     cx,1            ; col * 2
        add     ax,cx           ; column + (row * scrn width)
        mov     di,ax           ; point di to scrn
        cld                     ; direction increment
        mov     cx,plen         ; string length parameter
        mov     al,pattr        ; attribute to al
vdr1:
        inc     di              ; bypass character byte
        stosb                   ; al -> screen
        loop    vdr1
        pop     si
        pop     di
        ret
vdAttr  endp
        end

;/////////////////////////////////////////
```

Figure 13-17 *The source code for VDCHAR.ASM.*

```
;/////////////////////////////////////////
;//
;// vdchar.asm
;//
;// Write a screen token to the screen
;//
;// vdChar(int row, int col, int token);
;//
;/////////////////////////////////////////

        dosseg
IF   mdl eq 1
        .model small,c
ELSEIF  mdl eq 2
        .model medium,c
ELSE
        .model large,c
ENDIF

        extrn   SCRNSEG:word

        .code
```

```
        public   vdChar

vdChar  proc
        arg     prow:byte,pcol:byte,ptoken:word
        push    di
        push    si
        mov     cx,SCRNSEG
        mov     es,cx           ; reset extra seg
        xor     ax,ax           ; 0 -> ax
        mov     al,prow         ; row -> al
        mov     bl,160          ; 80 chars wide * 2
        mul     bl              ; row * scrn width  -> ax
        mov     cl,pcol         ; column to cl
        XOR     ch,ch           ; 0 -> ch
        shl     cx,1            ; col * 2
        add     ax,cx           ; column + (row * scrn width)
        mov     di,ax           ; point di to scrn
        mov     ax,ptoken       ; token to ax
        stosw                   ; ax -> screen
        pop     si
        pop     di
        ret
vdChar  endp
        end
;/////////////////////////////////////////
```

The source code for VDCHR.ASM.

Figure 13-18

```
;/////////////////////////////////////////
;//
;// vdchr.asm
;//
;// Write a character to the screen with no
;// change in the screen attribute
;//
;// vdChr(int row, int col, char ch);
;//
;/////////////////////////////////////////

        dosseg
IF   mdl eq 1
        .model small,c
ELSEIF  mdl eq 2
        .model medium,c
ELSE
        .model large,c
ENDIF

        extrn   SCRNSEG:word

        .code
```

Figure 13-18 *Continued.*

```
        public   vdChr

vdChr   proc
        arg     prow:byte,pcol:byte,pch:byte
        push    di
        push    si
        mov     cx,SCRNSEG          ; reset extra seg
        mov     es,cx               ; 0 -> ax
        xor     ax,ax               ; row -> al
        mov     al,prow             ; 80 chars wide * 2
        mov     bl,160              ; row * scrn width  -> ax
        mul     bl                  ; column to cl
        mov     cl,pcol             ; 0 -> ch
        XOR     ch,ch               ; col * 2
        shl     cx,1                ; column + (row * scrn width)
        add     ax,cx               ; point di to scrn
        mov     di,ax               ; char to ax
        mov     al,pch              ; ax -> screen
        stosb
        pop     si
        pop     di
        ret
vdChr   endp
        end
;////////////////////////////////////////
```

Figure 13-19 *The source code for VDWRITE.ASM.*

```
;////////////////////////////////////////
;//
;// vdwrite.asm
;//
;// Write a string the screen
;//
;// vdWrite(int row, int col, int length, char *str, int attr);
;//
;////////////////////////////////////////

        dosseg
IF   mdl eq 1
        .model small,c
ELSEIF  mdl eq 2
        .model medium,c
ELSE
        .model large,c
ENDIF

        extrn   SCRNSEG:word

        .code
        extrn   strlen:far
```

```
        public  vdWrite

vdWrite proc
        arg     prow:byte,pcol:byte,plen:word,pptr:ptr,pattr:byte
        push    di
        push    si
        push    ds
        mov     cx,dgroup:SCRNSEG
        mov     es,cx           ; reset extra seg
IF      mdl eq 3
        lds     si,[pptr]
ELSE
        mov     si,pptr         ; pointer to string
ENdiF
        xor     ax,ax           ; 0 -> ax
        mov     al,prow         ; row -> al
        mov     bl,160          ; 160 = (80 chars wide * 2)
        mul     bl              ; row * scrn width  -> ax
        mov     cl,pcol         ; column to cl
        xor     ch,ch           ; 0 -> ch
        shl     cx,1            ; col * 2
        add     ax,cx           ; column + (row * scrn width)
        mov     di,ax           ; point di to scrn
        cld                     ; direction increment
        mov     cx,plen         ; string length -> cx
        mov     ah,pattr        ; make word token
        or      cx,cx
        jnz     vdr1
vdr2:
        lodsb
        or      al,al
        jz      done
        stosw
        jmp     short vdr2
vdr1:
        lodsb                   ; get byte from string
        stosw                   ; token to screen
        loop    vdr1
done:
        pop     ds
        pop     si
        pop     di
        ret

vdWrite endp
        end

;////////////////////////////////////////
```

Epilogue

I cut my baby teeth writing 6502 assembly games for the Atari 800 computer. In a real sense, this book is a return to my programming roots. Seeing sprites, missiles, and bullets gliding about the screen, colliding, and then being reincarnated brought many smiles to my face.

It is my sincere hope that this book becomes a springboard for you being able to play with graphics and that it too brings smiles to your face.

I'd love to see any improvements in the sprite development system I've presented in this book, along with any games or graphic displays you'd like to share.

Namaste', Len
ldorfman@li.net

Index

A

animation
  bit-plane, 6-8
  technique from within sprite objects, 79-130
ASCII.H, 40-41

B

bit planes
  animating, 6-8
  animation library C++ code, 300-301
  definition/description, 4-5, 8
BITPLANE.LIB, 300-301
BLEEP.C, 342
Borland's graphical interface (BGI), 12
BOX.C, 22-29
BOXA.C, 55-77
bullet class, object manipulation in C++ code, 290-294
bullet objects, xi, 3-9
  simultaneous movement with sprites and missiles, 131-146
  source code, 146
BULLET.CPP, 290-294

D

DELAY.C, 346
Destory(...), 14
DSYRECT.C, 339
DTANK.C, 169-184

E

EXIT_BAD.C, 346

G

ghosts
  chasing game, 230-266
  detecting collisions with tanks and, 229
global data, C++ source code for, 299-300
GO.CPP, 299-300
GOCLASS.H, 29-32
GTKEY.ASM, 361-362
G_SHAPE.ASM, 362-363

H

hardware, requirements, xiii

K

key class, creating objects in C++ code, 297-298
KEY.CPP, 297-298
KEYBOARD.H, 36-41

L

LGHOST.C, 101-115
LTANK.C, 199-213

M

MASM, 132
missile class, object manipulation in C++ code, 285-289
missile objects, xi, 3-9
  colliding, 229
  launching from moving sprite, 215-225
  launching from tank, 227-266
  simultaneous movement with sprites and bullets, 131-146
  source code, 145-146
MISSILE.CPP, 285-289
MKATTR.ASM, 363
MKTOKEN.ASM, 364
MoveImage(...), 14
MSINIT.ASM, 365-366
MSOFF.ASM, 364-365
MSON.ASM, 366
MSSTAT.ASM, 366-367
MVCUR.ASM, 367-368

O

objects (*see* bullet objects; missile objects; sprite objects)
OFFCUR.C, 345
OFFSOUND.ASM, 368-369
ONCUR.C, 344
ONSOUND.ASM, 369
operators, RADIX, 8

P

PLAYF.CPP, 295-297
playfield, 7
  creating an instance of playfield class, 14
  destroying, 14-15
  moving a sprite object in, 11-46
  moving multicolored sprite objects on, 49
playfield class, creating objects in C++ code, 295-297
polling, 132-133

R

RADIX operator, 8
RCLOC.C, 352
RESTRECT.C, 341
RESTSCRN.C, 354
RGHOST.C, 116-130
RSIZECUR.C, 355-356
RTANK.C, 184-199

S

SAVERECT.C, 340-341
SAVESCRN.C, 353-354
SCLOC.C, 353
SCRNCLR.ASM, 370-371
SETRECT.C, 340
SIZECUR.C, 354-355
SIZERECT.C, 341-342
software, requirements, xiii
sound class, creating objects in C++ code, 298-299
SOUND.CPP, 298-299

source code
ASCII.H, 40-41
BITPLANE.LIB, 300-301
BLEEP.C, 342
Borland's sprite class library make file, 21-22
BOX.C, 22-29
BOXA.C, 55-77
bullet objects, 146
BULLET.CPP, 290-294
DELAY.C, 346
demo of keyboard-controlled sprite movement with missile launch, 217-225
demo of keyboard-controlled sprite movement, 149-154
demo of keyboard-controlled sprite movement/missile launch/chasing ghosts, 230-266
demo of moving one sprite object, 16-21
demo of moving one sprite/one missile/one bullet, 135-144
demo of moving two-color sprite objects with shape changes, 81-86
demo of simple movement of one sprite, 50-55
DSYRECT.C, 339
DTANK.C, 169-184
EXIT_BAD.C, 346
GO.CPP, 299-300
GOCLASS.H, 29-32
GTKEY.ASM, 361-362
G_SHAPE.ASM, 362-363
KEY.CPP, 297-298
KEYBOARD.H, 36-41
LGHOST.C, 101-115
LTANK.C, 199-213
missile objects, 145-146
MISSILE.CPP, 285-289
MKATTR.ASM, 363
MKTOKEN.ASM, 364
MSINIT.ASM, 365-366
MSOFF.ASM, 364-365
MSON.ASM, 366
MSSTAT.ASM, 366-367
MVCUR.ASM, 367-368
OFFCUR.C, 345
OFFSOUND.ASM, 368-369
ONCUR.C, 344
ONSOUND.ASM, 369
PLAYF.CPP, 295-297
PROG2-1.cpp, 16-21
PROG3-1.CPP, 50-55
PROG4-1.CPP, 81-86
PROG5-1.CPP, 135-144
PROG6-1.CPP, 149-213
PROG7-1.CPP, 217-225
PROG8-1.CPP, 230-266
RCLOC.C, 352
RESTRECT.C, 341
RESTSCRN.C, 354
RGHOST.C, 116-130
RSIZECUR.C, 355-356
RTANK.C, 184-199
SAVERECT.C, 340-341
SAVESCRN.C, 353-354
SCLOC.C, 353
SCRNCLR.ASM, 370-371
SETRECT.C, 340
SIZECUR.C, 354-355
SIZERECT.C, 341-342
SOUND.CPP, 298-299
sprite editor make file, 333-336
SPRITE.C, 310-333
SPRITE.CPP, 278-285
SPRITE_A.ASM, 270-273
SSIZECUR.C, 355
S_SHAPE.ASM, 370
TPROTO.H, 33-35
TSTRUCT.H, 41-45
UGHOST.C, 86-101
UTANK.C, 155-169
VDATTR.ASM, 371-372
VDBOX.C, 342-344
VDCHAR.ASM, 372-373
VDCHR.ASM, 373-374
VDEDIT.C, 347-352
VDWRITE.ASM, 374-375
VIDEO.CPP, 294-295
VIDINIT.ASM, 359-361
VRDCHAR.C, 345

sprite class
class library in assembly language code, 270-273
object manipulation in C++ code, 278-285
sprite editor, 305-336
make file, 333-336
source code, 310-336, 339-356, 357-375
support functions in assembly language, 357-375
support functions in C++, 338-356
using, 306-309
sprite objects, xi, 3-9
animation technique from within, 79-130
changing shape as direction changes, 80
controlling with a track, 147-213
designing, 8
including in program source code, 13
initiating sprite class, 13-14
launching missile from moving, 215-225
moving, 16-21
moving in playfield space, 11-46
moving multicolored, 47-77
moving simultaneously with missiles and bullets, 131-146
source code, 21-45, 50-77, 81-130, 135-146, 149-213, 217-225, 230-266
three-colored, 47-77
SPRITE.C, 310-333
SPRITE.CPP, 278-285
SPRITE_A.ASM, 270-273
SSIZECUR.C, 355
S_SHAPE.ASM, 370

T

tank game, 227-266
framework, 228-229
TASM, 132
TPROTO.H, 33-35
track, controlling sprite objects within a, 147-213
TSTRUCT.H, 41-45

U

UGHOST.C, 86-101
UTANK.C, 155-169

V

VDATTR.ASM, 371-372
VDBOX.C, 342-344
VDCHAR.ASM, 372-373
VDCHR.ASM, 373-374
VDEDIT.C, 347-352
VDWRITE.ASM, 374-375
video class, creating objects in C++ code, 294-295
VIDEO.CPP, 294-295
video graphics adapter (VGA), 4-5
VIDINIT.ASM, 359-361
VRDCHAR.C, 345

About the author

An educator and programmer, Len Dorman, Ph.D., is author or coauthor of many programming books, including *C++ Memory Management*, *Cross-Platform Development in OS/2*, *Effective Multithreading in OS/2*, and *Lightning-Fast Animation Graphics*. He is also a contributing writer to *OS/2 Professional* magazine and speaks at OS/2 user group meetings. He is CEO of MultiGrain Solutions, a software development and consulting company that integrates weak artificial intelligence technology into commercial applications.

DISK WARRANTY

This software is protected by both United States copyright law and international copyright treaty provision. You must treat this software just like a book, except that you may copy it into a computer in order to be used and you may make archival copies of the software for the sole purpose of backing up our software and protecting your investment from loss.

By saying "just like a book," McGraw-Hill means, for example, that this software may be used by any number of people and may be freely moved from one computer location to another, so long as there is no possibility of its being used at one location or on one computer while it also is being used at another. Just as a book cannot be read by two different people in two different places at the same time, neither can the software be used by two different people in two different places at the same time (unless, of course, McGraw-Hill's copyright is being violated).

LIMITED WARRANTY

McGraw-Hill takes great care to provide you with top-quality software, thoroughly checked to prevent virus infections. McGraw-Hill warrants the physical diskette(s) contained herein to be free of defects in materials and workmanship for a period of sixty days from the purchase date. If McGraw-Hill receives written notification within the warranty period of defects in materials or workmanship, and such notification is determined by McGraw-Hill to be correct, McGraw-Hill will replace the defective diskette(s). Send requests to:

McGraw-Hill, Inc.
Customer Services
P.O. Box 545
Blacklick, OH 43004-0545

The entire and exclusive liability and remedy for breach of this Limited Warranty shall be limited to replacement of defective diskette(s) and shall not include or extend to any claim for or right to cover any other damages, including but not limited to, loss of profit, data, or use of the software, or special, incidental, or consequential damages or other similar claims, even if McGraw-Hill has been specifically advised of the possibility of such damages. In no event will McGraw-Hill's liability for any damages to you or any other person ever exceed the lower of suggested list price or actual price paid for the license to use the software, regardless of any form of the claim.

MCGRAW-HILL, INC. SPECIFICALLY DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Specifically, McGraw-Hill makes no representation or warranty that the software is fit for any particular purpose and any implied warranty of merchantability is limited to the sixty-day duration of the Limited Warranty covering the physical diskette(s) only (and not the software) and is otherwise expressly and specifically disclaimed.

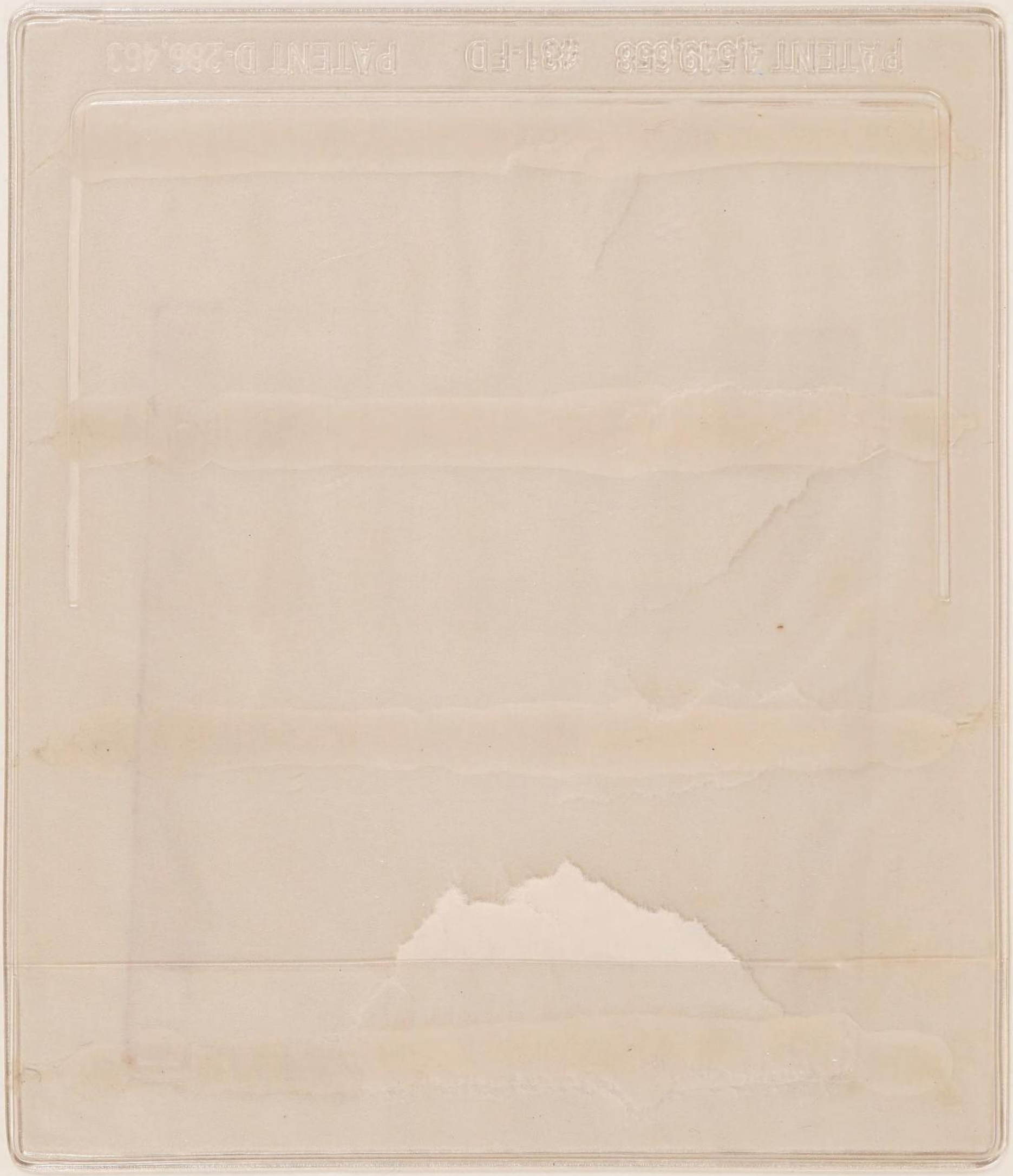
This limited warranty gives you specific legal rights; you may have others which may vary from state to state. Some states do not allow the exclusion of incidental or consequential damages, or the limitation on how long an implied warranty lasts, so some of the above may not apply to you.

About the enclosed disk . . .

This disk contains a readme file and two self-extracting archive files. Just create the subdirectories \EDITOR and \SOURCE on your hard drive, copy the compressed files (EDITOR.EXE and SOURCE.EXE) to the appropriate subdirectory, and then type the name of the file at the \EDITOR> or \SOURCE> prompt. Once the files are expanded, \SOURCE will contain all the source code for the *Game Graphics in C++* animation class library, along with all the source code for the demonstration programs, and \EDITOR will contain all the source code for the sprite editor program.

IMPORTANT

Read the Disk Warranty terms on the previous page before opening the disk envelope. Opening the envelope constitutes acceptance of these terms and renders this entire book-disk package nonreturnable except for replacement in kind due to material defect.


PATENT 4,519,658 #81-FD PATENT D-286,463

$34.95 U.S.A.

Programming

Animate missiles, bullets, and multicolored sprites using C++!

For both C and C++ programmers, this book/disk package gives you reusable graphic objects, including a C++ library that simplifies bit-plane animation implementation in DOS-based programs.

C++ expert Len Dorfman shows you how to incorporate high-speed sprite, missile, and bullet movements into your programming arsenal, covering important techniques such as polling and collision detection. He doesn't waste time covering material you don't need to know, instead zeroing in on practical applications.

This time-saving guide is divided into three sections:

- **Part I** — The demonstration programs
- **Part II** — C++ graphic object source code for the bit-plane animation class library
- **Part III** — The sprite editor, with C and assembly source code

With the complete documentation provided here, you'll learn how to harness the OOP features of C++ to jazz up your own graphics programming—without wading through a lot of jargon.

DISK Included

Disk includes:

- *7 fully documented C++ animation demo programs*
- *Complete object-oriented C++ graphics library*

Up to date for the latest version of Borland C++

Compatible with DOS-equipped PCs with VGA graphics board

P/N 017946-8
PART OF
ISBN 0-07-911951-4
90000
9 780079 119513

Cover: Square One Design, Albuquerque, NM
Cover Photo: Marjory Dressler, New York, NY

9780079119513
02/27/2018 8:38-2
22